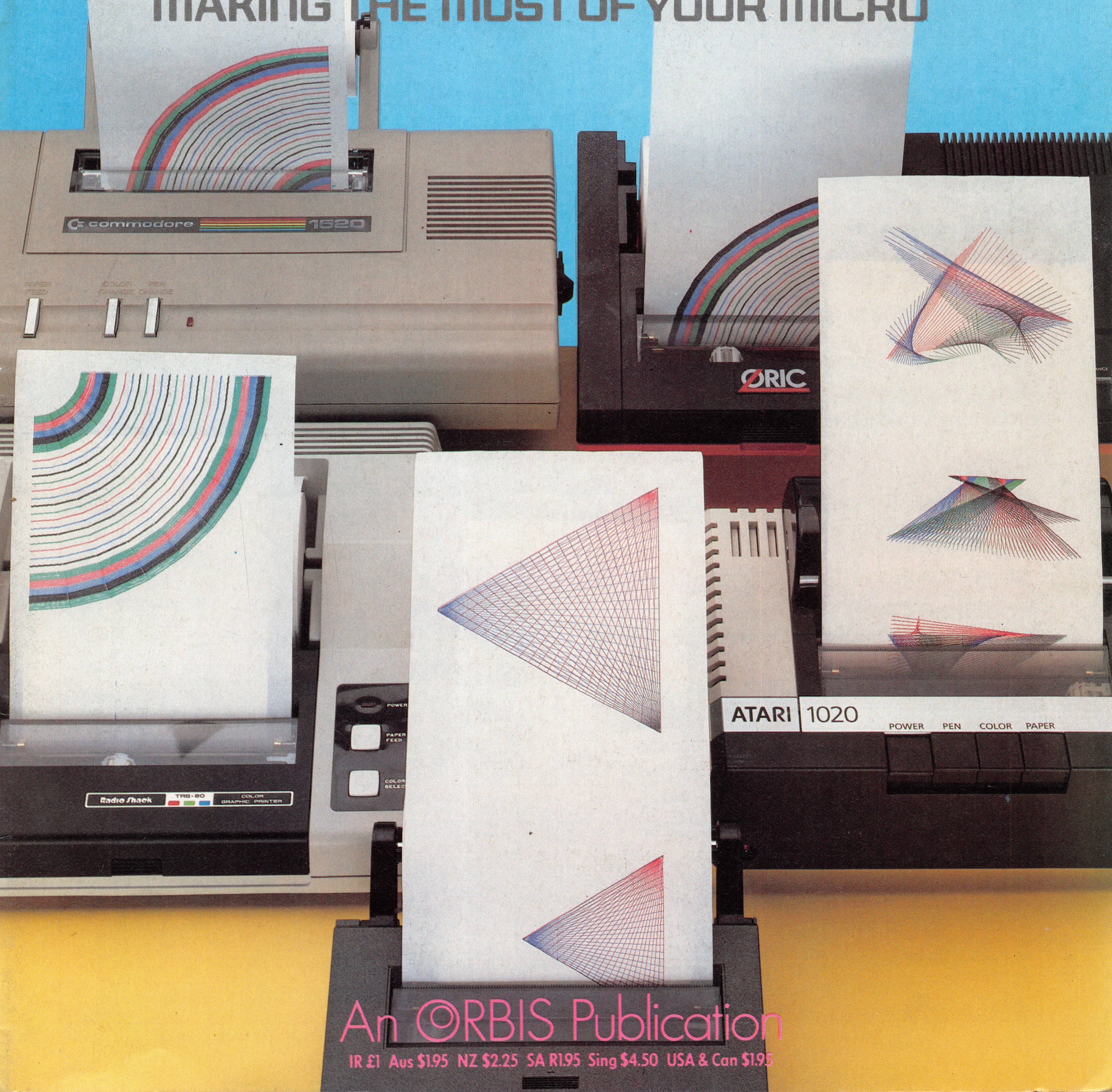
ISSN 0265-2919
80p 15

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication
IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- We look at Eaca's Colour Genie, a home computer with lots of graphics facilities, a built-in tape loading meter and joysticks with numbers keys on them
- We look at the principles behind some of the chess programs available for home computers
- We take an overview of the range of spreadsheet programs available for micros

QUIZ

1) What happens to the registers V+6 and V+16 when X crosses the page boundary?

2) When SAVEing a file to tape, what information would be contained in the first record, A$(0,0)?

3) Why is a membership number on a database stored in a character field and not a number field?

4) The plotting range on a printer/plotter is +999 to −999 for both the X and Y directions. However, in practice the X axis is limited. Why is this, and what is the actual range?

Answers To Last Week's Quiz

A1) Z80 registers have the ability to address 16-bit addresses.

A2) The letter Z will appear on the screen as the function key has no effect at all.

A3) None. You will lose a life if you touch an electric eel.

A4) To prevent the string variable TI$ from overwriting the area devoted to sprite data.

QUIZ

COVER PHOTOGRAPHY BY CHRIS STEVENS

Editor Jim Lennox; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Picture Editor** Claudia Zeff; **Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Contributors** Steven Colwill, Max Phillips, Matt Nicholson, Rose Deacon, Richard Pawson; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1P 1LB; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95
How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.
Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.
How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.
Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

FACTS AT YOUR FINGERTIPS

Computing is all about the storage and processing of information. One of the most important types of software for storing data is the database. We take a detailed look at the uses to which databases are put on home micros, pointing out the limitations of using them on the smaller machines and showing how they are used to best advantage.

A *database* is a collection of data about one subject. For example, a database could hold one of the following sets of information: the names and addresses of members of a club; the expenses incurred by club meetings; the dates and venues of club meetings; or the payment of club fees by the membership. However, a database system could just as easily include all of these sets of data in one large file.

A *file* is a collection of records, each of which comprises a number of fields. In some applications, such as accounts packages, the structure of the file is fixed by the software, but in a database system it is more usual for the file layout to be chosen to suit each task. This involves determining the size of each field and defining the type of contents to be stored in it. In a name and address file, for example, each person's details occupy an individual record: the name is stored in one *character field* and each line of the address in separate fields. Such fields are usually thirty characters long, allowing no more than thirty letters in each.

The alternative to a character field is a *number field*, which allows the program to do calculations on the data stored in it and thus produce useful reports. In our club membership file, the program might calculate how many members have paid their fees, what the total income has been for the year so far, or how much is still to be collected. Not all numerical data, however, needs to be stored in a number field. Telephone numbers are a good example of those on which calculations are never performed, and these are stored in character fields.

Before you can tell whether a file could effectively be used on your computer, you need first to estimate the maximum size of the file (how many records you will want to keep), calculate the amount of memory needed for each record, and then determine whether you have enough memory available to store the file. A record with a name and three address fields of 30 characters each and a telephone number taking 10 characters occupies a total of 130 bytes. If you have a disk

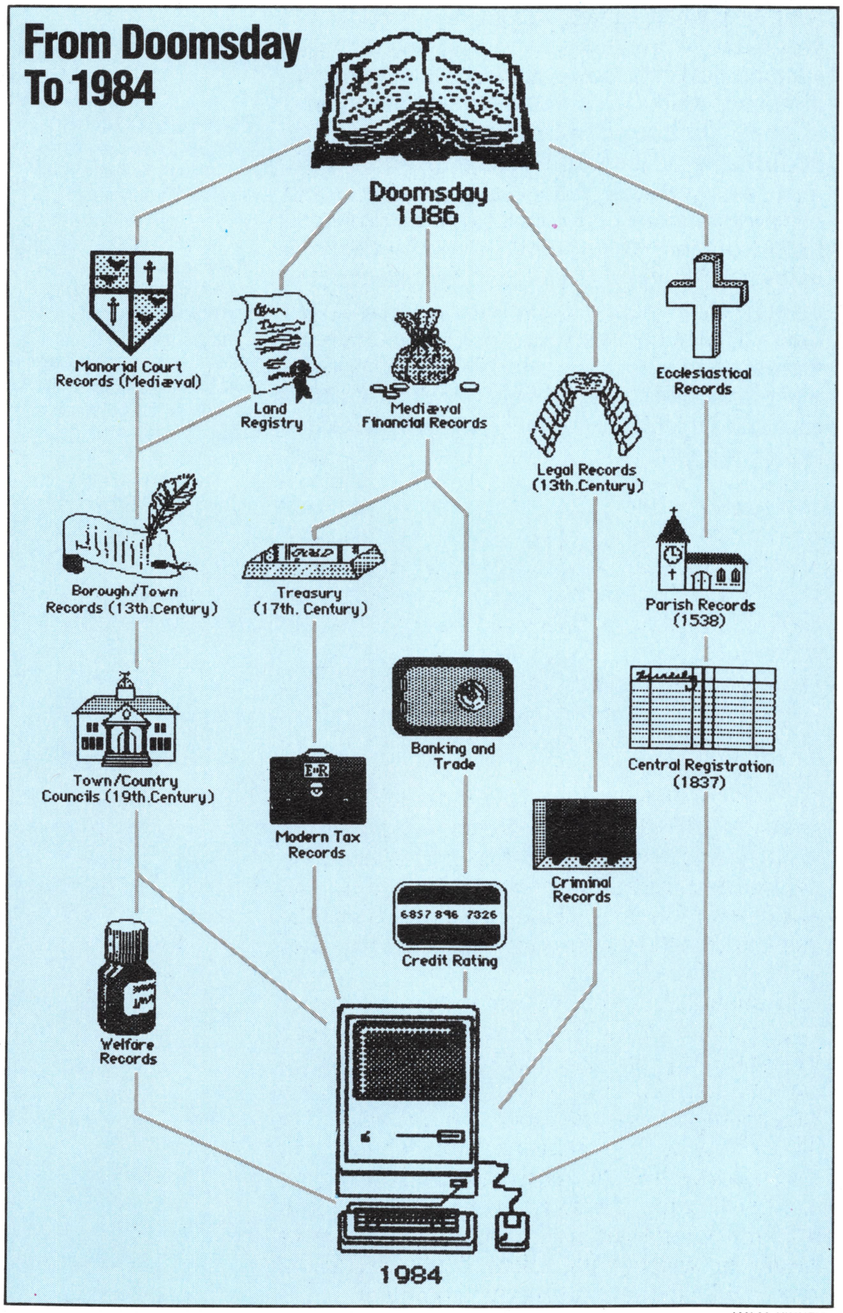
Collecting One's Thoughts
Most people keep personal information about themselves unsystematically filed on scattered pieces of paper. A home microcomputer database might be useful as a central store of insurance records, property details, bank account numbers, etc. Furthermore, it should be invaluable to anyone who collects stamps, coins, records, or any collection that lends itself to systematic classification

MIKE BROWNLOW

Only Connect
Databases proliferate in technical societies that depend upon the accumulation of facts, and the accessibility of information grows as technology progresses. In the past, databases have been isolated from other databases by distance and time, but now that so many governmental and private databases are computer-based the possibility of their communicating with each other may threaten privacy and confidentiality

system with 200 Kbytes storage per disk then you could hold 1,500 records on each floppy. In a tape-based system with 48 Kbytes of memory you would probably be able to have only 300 records (allowing about 10 Kbytes for the program and operating system). In practice, if you wish to sort the file into order or do other such manipulations, you will need some spare working space, and it is wise to limit the size of file to half the available area. The two storage systems differ so drastically because tape files must be read into memory and processed as a whole, whereas the speed of disk file accesses allows files to reside on disk and be processed in memory.

A database management system would enable you to take information from one file — the total expenses for the year, for instance — and link or compare this with information from another file, such as the total income from membership fees.



INFORMATION FROM HEAD OF RESEARCH, PUBLIC RECORDS OFFICE

IAN McKINNELL

You could then take an informed decision about whether to start a recruitment campaign, cut down the number of meetings or spend the profits on a new computer for the club. You might wish to send a standard letter to club members informing them of the situation. The database could provide the names and addresses and print them straight onto envelopes or sticky labels. Alternatively, it could link with a word processing package to write letters and reports. Although there are many things that the ideal database can do, you will not easily find one that provides all the facilities together, particularly if you have a home computer with restricted available memory space and tape rather than disk storage. Many of the less sophisticated database packages handle only one file at a time, so that although you could hold all the different sets of information described above, you could not link or compare them in true database fashion.

A real danger in all data processing is putting all of your precious information into a computer and then through some accident obliterating it. It is essential with important information to keep security copies and also to take a print-out at intervals so that nothing can be irretrievably lost.

A simple database package will handle a single file of information. The system should make the routine work of entering data into the file as easy as filling out a form, and it should allow you to access information both on the screen and to a printer. The user should be able to select individual records by stating search and sort criteria such as 'all members who have not paid this year and have a given postcode'. Such a package should also be able to print out specific record fields, such as the name and address fields alone for mailing lists. This simple kind of package is available for most tape-based home computers. If you have a disk system, however, the range of software available is much greater.

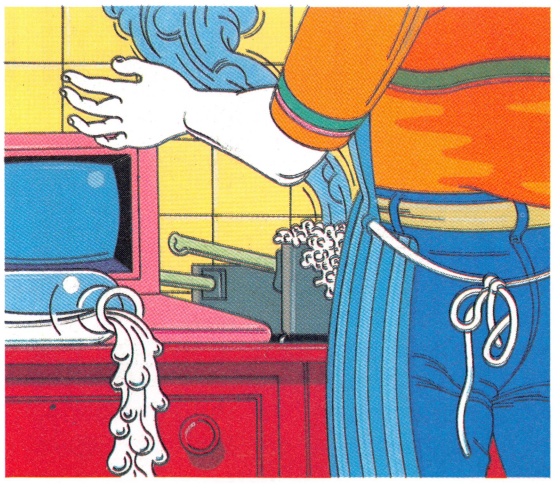
Apart from the uses already suggested, database systems on home computers can be used to catalogue record collections, books, bibliographical references and stamp collections, or keep records of cricket or football team scores. More specialised information-handling tasks include second-hand book collecting and the organisation of self-help baby-sitting groups.

Slightly more sophisticated packages will allow you to design a data entry form on the screen to match a previous paper-based system. Other packages will let you select numerical items that are greater or smaller than a stated amount (such as all members owing more than £10). It is therefore important to shop around for the package that best suits your specific needs.

A computer with a database package is useful for tasks that involve repetition, or where a lot of information needs to be sorted or searched through quickly. But some applications are not at all suitable for database systems. It is not sensible to search through something like a file of telephone numbers on a home computer every

A Recipe For Disaster

Eric Nowhere has a Vic-20 that he is extremely fond of. It has helped him to come to terms with computers and learn a little about programming in BASIC, but Eric unfortunately cannot distinguish between a serious application and having fun. He insists on using the computer to keep the telephone numbers of his (few) friends, and he uses a database to store his cooking recipes. When you go to supper at his house, you invariably get nothing to eat till about midnight because Eric insists on checking out the details of the cooking instructions on his Vic. He plugs it in with a dramatic flourish, loads the program in from tape — this usually takes several attempts — waits happily during the interminable period while the computer searches and reads, views the instructions a few lines at a time on the screen, and finally heads off to the kitchen to cook. You quickly unplug the computer and switch on the television, knowing you are in for a long hungry wait. It's no good coming to supper late as he won't start before you get there, because part of the excitement for him is to show you the computer at work. Eric clearly doesn't make efficient use of his computer.



ROY INGRAM

The Right Records

Peggy Average has always had an enormous record collection and has built up a profitable disco service. Experience has taught her that it is essential to take the right records for each type of gig. Recently, Peggy decided to buy a computer, and for speed and high storage capacity she chose a twin-disk system. She then worked out on paper the appropriate headings for the categories of music that she wanted her database program to include. There were two main groups, each with subheadings that were further subdivided. The main division was between 'Black' and 'White' music. Black music consisted of three main sub-groups: Reggae, Jazz and Soul. White music broke down into Rock and New Wave. White Rock subdivided into Heavy Metal and Progressive Rock, and so on. Peggy used the database manager to pick records according to the occasion — entering, for instance, the command to list all Rock records and then narrowing down the choice by making a lower-level selection of either Heavy Metal or Progressive Rock. Having inspected the listing of record titles displayed on the monitor she took a print-out of the titles in the required target area and then went to the shelves and took off the ones she wanted for that occasion.

time you wish to make a telephone call. In the time it takes you to switch on the computer, load up the database program, call the file and sort through it for Lizzie Dixon's number, you could have made six telephone calls to numbers obtained from a book or card index. You might want to keep the names and telephone numbers on file for some other purpose, such as printing standard letters and labels, but using a computer system when established manual systems are perfectly adequate is pointless.

A more ambitious database package will provide all of these facilities, and will give you the ability to use arithmetic to obtain such data as the total amount of revenue earned from club fees, or calculate which club facility is used most often. Several files of information can be linked so that data on related subjects can be used in conjunction with one another. For example, a particular player belonging to our imaginary club has his personal details kept in one file and details of his performance in club tournaments in another. When a history of his 'form' is required, the two files can be used together. For a 'snapshot' of the performance of everybody in the club on a particular date, the file can be accessed by calling up every record for a certain date, rather than every record for a certain person. In order to run a program with such extensive facilities, it would almost certainly be necessary to use a desktop or business computer with a minimum of 64 Kbytes of memory and disk drives.

There has been a remarkable increase in the use of information systems in the last few years. Doctors use databases for keeping details of their patients' medical histories. Research workers use them for specialised sorting and cross-referencing of data. Businesses keep mailing lists and customer information handy and up to date. The more information there is available, the more they are used. Databases can now handle any form of information, from simple lists to highly complex systems with multiple files, facilities for creating elaborate reports and the ability to process information in a form that can be transferred to other computer software such as word processors. The ability to communicate with public-access databases such as Prestel and Micronet 800 gives the home computer user access to an enormous amount of information. This points the way to a future in which your micro could become a terminal linked to a mainframe computer.

UNDERWATER SPRITES

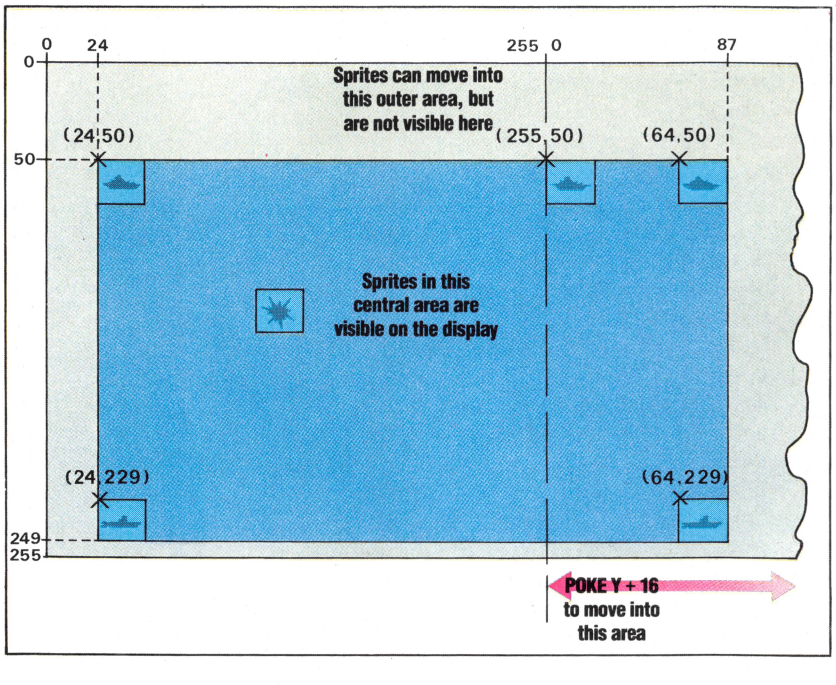
Our series on Commodore 64 graphics is now reaching its final stages. Having examined the mechanics of designing, colouring and expanding sprites, we now turn our attention to sprite movement. The routines that control the ship, submarine and depth charges in our Subhunter game are looked at in detail here.

To position a sprite on the screen it is necessary to specify an X and a Y co-ordinate. Each sprite is defined on a 21 × 24 pixel grid, and the co-ordinates are measured from the *top left-hand corner* of the grid. Co-ordinates are held in registers within the VIC chip. These are assigned as follows:

Sprite Number	**0**		**1**		**2**		**3**		**7**	
Co-ordinate	X	Y	X	Y	X	Y	X	Y	X	Y
VIC Register	V	V+1	V+2	V+3	V+4	V+5	V+6	V+7	V+14	V+15

Each location can accept numbers in the range 0 to 255. This is more than sufficient to specify one of the 200 pixels in the vertical (Y) direction, and the extra capacity is used to allow a sprite to move in and out of view at the top or bottom of the screen. However, the 320 pixels in the horizontal (X) direction exceed the maximum of 255 allowed by an eight-bit register, so a further bit is allocated to each sprite. These extra bits are grouped together in a single register with the address V+16. The diagram below shows the screen limits for fully visible, unexpanded sprites.

The Sprite Position
This diagram shows the parameters for positioning sprites on the screen of the Commodore 64. Placed in locations within the central part of this screen, a sprite will be visible on-screen. If a sprite is moved to the outer part of the display then it will not be seen. The four sprites in the corner of the central area show the limits of how far a sprite can be moved and still be entirely visible. The co-ordinates of the top left hand corners of the sprites are given



0
24
255 0
87
0
50
249
255
Sprites can move into this outer area, but are not visible here
(24,50)
(255,50)
(64,50)
Sprites in this central area are visible on the display
(24,229)
(64,229)
POKE Y + 16 to move into this area
KEVIN JONES

The ship's movement is controlled by program lines 230-250 and 270-290. Initial co-ordinates for the ship were set up in line 2270 of the sprite creation routine (see page 265). The ship will move in a horizontal direction only, so the Y co-ordinate will remain fixed. If this is set to 80, the ship will be correctly positioned on the ocean, which was designed using the screen setup routine. The ship's X co-ordinate, X0, is initially set to 160, giving a central starting position.

The ship is controlled from the keyboard, using the 'Z' and 'X' keys for left and right movement. Lines 230-250 of the main loop GET a character from the keyboard. If no key is pressed, program execution continues (in contrast to the INPUT command, which halts a program until the key is pressed). The value obtained by GET is then stored in A$ and used to modify the ship's X co-ordinate: if the 'Z' key is pressed, the ship is moved a short distance to the left and the X co-ordinate is decreased; if 'X' is hit it will be moved the same distance to the right and the X co-ordinate will be increased. The second part of lines 240 and 250 contains a test to see if the ship has reached the limits of its travel. The IF...THEN structure in Commodore BASIC is such that if the first condition on a program line is untrue then the rest of the line is not executed. In our program, condition testing is arranged so that computer time is not wasted in needless calculation. For example, there is no need to test for the lower limit of X0 if the 'X' key is being pressed and the value of X0 is increasing. After testing, the value of X0 is POKEd into location V, the X co-ordinate register for sprite 0.

The movement of the submarine is controlled by lines 300-350 and 2500-2570, and operates within the main loop of the program as shown in the flow chart opposite. The 'Reset Sub Co-ords' subroutine starting at line 2500 uses the RND function to select the sub's depth, Y3, and speed, DX. Y3 will always be an integer number in the range 110 to 214, thus ensuring that the sub will not appear above the sea surface or below the seabed. The speed factor, DX, falls in the range 1 to 3 and sets the number of pixels by which the sub's X co-ordinate will be increased or decreased. The sub's X co-ordinate and the V+16 register bit that controls sprite 3's X co-ordinates above 255 are both set to zero.

In lines 300-350 of the main loop the value of DX selected is added to the sub's X co-ordinate and this is then tested to see if the sub has reached the edge of the screen. Lines 340 and 350 allow us to deal with the problem of having values of X3 in excess of 255. As the X co-ordinate increases to, say, 256, two things must happen — the bit corresponding to sprite 3 in the V+16 register is set to one and the normal X co-ordinate register starts again from zero. The following table shows what happens in the registers as the submarine crosses the X = 255 boundary:

X3	V+16								V+6							
254	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
255	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
256	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
257	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1

The variable H3 will be set to one if the value of X3 exceeds 255. Correspondingly, L3 will be reset to zero if H3 becomes one. The values of L3 and H3 can then be POKEd into the registers V+6 and V+16.

FIRING THE DEPTH CHARGES

During the game, depth charges can be dropped on the submarine at any time. To make a program straightforward, we shall make the rule that once a depth charge has been fired no other depth charge can be fired until:

a) the sub has been hit; or

b) the charges have missed the sub and dropped a little way past it; or

c) the charges have missed the sub and reached the seabed.

The main loop of the program has two jobs to do in respect of depth charges: it must detect the pressing of the 'M' key, and once the depth charge has been fired it must control its vertical movement. The program must also ensure that no new depth charges are fired whilst one is in the process of dropping. This last problem can be solved by the use of a flag. This is a technique often applied in program control, signalling that a particular event has, or has not, occurred. In our program we shall use the variable FL to signal the dropping of a depth charge. Its value will be one if a charge is dropping and zero otherwise. In line 100 of the program the value of FL is initially set to zero. Line 260 accesses the 'Set Up Depth Charges' subroutine at line 3000 if 'M' is pressed and the flag is set to zero. A second subroutine at line 400 is used to move a fired depth charge sprite, and this is accessed by line 380.

The 'Set Up Depth Charges' subroutine has three functions to perform:

1) To set the flag FL to one as a signal that a charge has been fired.

2) To set the starting co-ordinates: the X co-ordinate takes its value from that of the ship and the Y co-ordinate is initially set to 95, positioning the charge just below the surface of the sea.

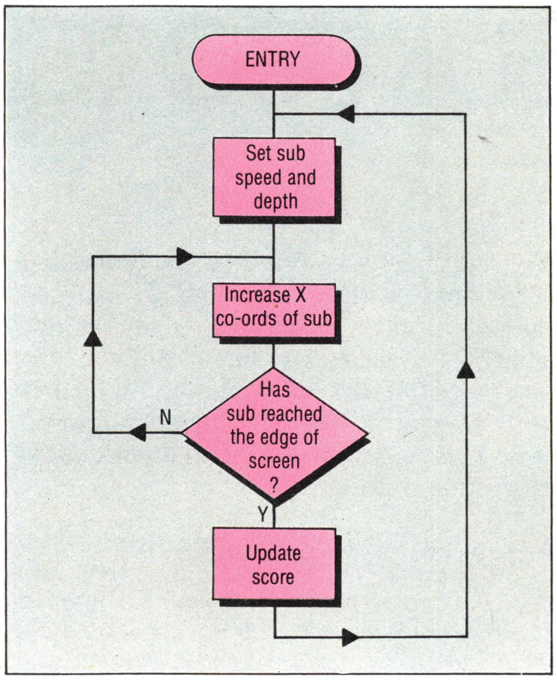
3) To turn on the depth charge sprite.

The 'Move Depth Charge' subroutine is used to move the depth charge down the screen. In addition, tests have to be made to see if:

1) The depth charge has gone past the submarine *or* reached the seabed.

2) The depth charge has hit the sub.

If the first event has occurred, the depth charge sprite can be turned off and the flag reset to zero,

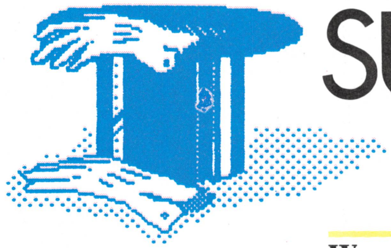

ENTRY
Set sub speed and depth
Increase X co-ords of sub
Has sub reached the edge of screen ?
N
Y
Update score
KEVIN JONES

Underwater Flow Chart
This simple flow chart shows how the program controls the movement of the submarine sprite. It selects a random depth and speed, being careful to make sure that the sub is under the surface and above the seabed. The sub is then moved smoothly across the screen until it reaches the other side

allowing another depth charge to be fired. The second event is tested by using another feature of Commodore 64 sprites — the sprite collision register. As with other registers of the VIC chip, this register, V+30, has one bit corresponding to each sprite. If a particular sprite is involved in a collision with another sprite then the corresponding bit in this register is set to one. Thus, if the sub (sprite 3) and the depth charge (sprite 2) are in collision, the contents of the register V+30 will be 12 (00001100 = 12). By PEEKing this register and testing its contents, we can tell if the depth charge has hit the sub. If it has, then a further 'HIT' subroutine is accessed at line 5000. This subroutine will be dealt with in the final instalment of the project, together with the instructions to update the HI SCORE and restart the game.

Subhunter's Movement Subroutines

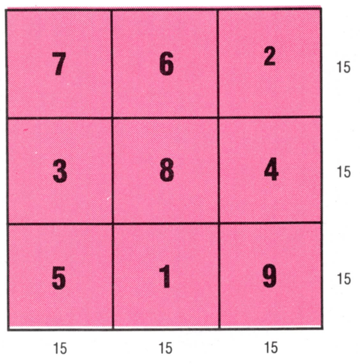
```
130 GOSUB 2500: REM SET SUB CO-ORDS
230 GET A$
240 IF A$="Z" THEN X0=X0-1.5: IF X0<24 THEN
X0=24
250 IF A$="X" THEN X0=X0+1.5: IF X0>245 THEN
X0=245
260 IF A$="M" AND FL=0 THEN GOSUB 3000:
    REM SET DEPTH CHARGES
265 :
270 REM ** MOVE SHIP **
290 POKE V,X0
295 :
300 REM ** MOVE SUB **
310 X3=X3+DX
315 :
320 REM  ** SUB AT SCREEN END? **
330 IF X3>360 THEN DS=-1:GOSUB 5500:
GOSUB 2500
340 H3=INT(X3/256):L3=X3-256*H3
350 POKE V+6,L3: POKEV+16,PEEK(V+16)OR(8*H3)
360 IF FL=1 THEN GOSUB 4000:
 REM MOVE DEPTH CHARGE
370 GOTO 200: REM RESTART MAIN LOOP
380:
390:
2500 REM **** RESET SUB COORDS ****
2510 Y3=110+INT(RND(TI)*105)
2520 X3=0:DX=RND(TI)*3+1
2530 POKE V+7,Y3:POKE V+6,X3
2540 POKE V+16,PEEK(V+16) AND 247
2550 RETURN
2560 :
2570 :
3000 REM ***SET UP DEPTH CHARGES***
3010:
3020 REM ** SET FLAG **
3030 FL=1
3040 :
3050 REM ** SET COORDS ***
3060 Y2=95:X2=X0
3070 POKE V+4,X2:POKE V+5,Y2
3080 :
3090 REM ** TURN ON SPRITE 2 **
3100 POKE V+21,PEEK(V+21)OR4
3110 RETURN
3120 :
3130 :
4000 REM ** MOVE DEPTH CHARGE **
4010 :
4020 REM ** INCREASE Y COORD **
4030 Y2=Y2+2
4040 POKE V+5,Y2
4050 :
4060 REM **TEST BOTTOM & TURN OFF**
4070 IF Y2>Y3+25 OR Y2>216 THEN
POKE V+21,PEEK(V+21) AND 251:FL=0
4080 :
4090 REM **TEST FOR SUB HIT**
4100 IF PEEK(V+30)=12 THEN GOSUB 5000:
REM HIT ROUTINE
4110 RETURN
4120 :
4130 :
```



SUM SORCERY

We continue our series of programming suggestions with a look at an age-old mathematical brain-teaser — the 'magic square'. This puzzle is an ideal candidate for experimentation: the solution is fairly straightforward, but its implementation on home computers involves us in solving some interesting problems.

A magic square is simply a grid of cells into which the positive integer numbers (1, 2, 3, 4, 5, etc.) are entered (zero is not allowed and no number should be used twice). The object of the exercise is to distribute these numbers in such a way that each column and each row will give the same total when the numbers are added together. The simplest magic square is a three-by-three box, and one possible solution is:

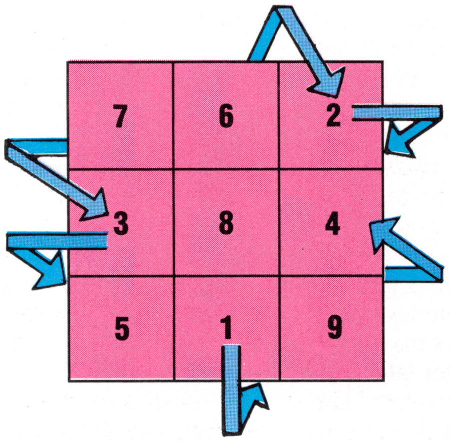


In this example, all the rows and columns add up to 15. As an exercise, try the puzzle yourself — but this time use a five-by-five grid and enter the numbers from 1 to 25. You will find that this isn't as easy as it looks, and undoubtedly considerable trial and error will be needed before the correct solution is obtained.

Fortunately, your computer can make things easier. One answer is to write a program to fill the squares with numbers and then test each row and column individually. However, this method could well take many hours to find a solution for even quite small squares, seven or nine cells wide. What is needed is a method for generating the numbers. To save you the trouble of discovering your own, here's one that works every time:

1) Start with the number 1 in the middle cell on the bottom row.
2) After each cell is filled, move one cell down and one cell right before entering the next number. If moving right takes you off the right-hand edge of the grid, go to the first column instead. Similarly, if moving down takes you off the bottom of the grid then move to the top row. This is similar to a screen 'wraparound' effect.
3) If the next cell is already occupied, move one position to the left of the last cell used.
4) Carry on until you reach the last empty cell.

Applying this method to our example shows how a three-by-three square is generated:

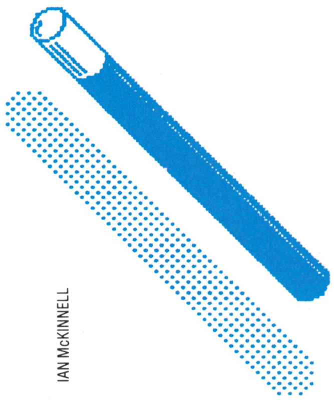


It's very easy to write a program to do this for any size of square. Simply create an empty two-dimensional array of the correct size for the desired magic square, and then use a loop to fill it in according to the rules given. You should note that the method works only for magic squares with

Squares of Enchantment

These squares were generated by the program and have been successfully checked by it — as the printed message shows. In the 15 × 15 square, a diagonal pattern of three- and two-digit numbers can be observed, the consequence of the construction algorithm

```
52  42  32  22  12   2  73  72  62
63  53  43  33  23  13   3  74  64
65  55  54  44  34  24  14   4  75
76  66  56  46  45  35  25  15   5
 6  77  67  57  47  37  36  26  16
17   7  78  68  58  48  38  28  27
19  18   8  79  69  59  49  39  29
30  20  10   9  80  70  60  50  40
41  31  21  11   1  81  71  61  51
```



IAN McKINNELL

an odd number of cells — your program should reject even-numbered grids.

Care should be taken when the square is displayed. In the interest of neatness, all numbers should be aligned correctly in rows and columns. This is achieved quite simply if your micro features the PRINT USING command. If not, it is best to convert the number to be printed into a string. This can then be padded out with space characters so that the 'number' is always the same length. A subroutine to achieve this is:

```
1000 REM Convert A to A$ and align
1010 A$=STR$(A)
1020 IF LEN(A$)<3 THEN A$=" "+A$:GOTO 1020
1030 RETURN
```

The exact method, of course, will depend on the particular computer being used.

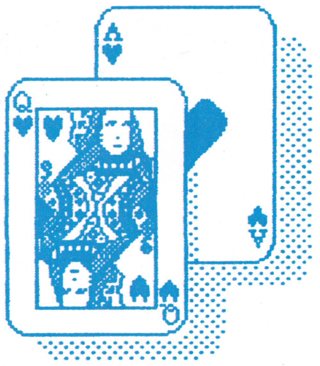
The next problem is the screen size — most micros will be unable to display large magic squares on-screen. A 40-column screen has room for 13 two-digit columns, but a 13-by-13 square will include some three-digit numbers, so a nine-by-nine square is the largest that may be displayed. A printer will allow much larger squares to be generated. Most printers have a maximum width of 80 or 132 columns, and larger squares may be printed in sections that are joined together later.

The overall aim of this project is to create the largest magic square you can, and present it as neatly as possible. As an experiment, try writing a trial and error program and determine how long it takes to find an answer (although bear in mind that this will take a ridiculously long time to achieve a result). Alternatively, you could look for even faster methods for generating the squares.

```
130 114  98  82  66  50  34  18   2 211 210 194 178 162 146
147 131 115  99  83  67  51  35  19   3 212 196 195 179 163
164 148 132 116 100  84  68  52  36  20   4 213 197 181 180
166 165 149 133 117 101  85  69  53  37  21   5 214 198 182
183 167 151 150 134 118 102  86  70  54  38  22   6 215 199
200 184 168 152 136 135 119 103  87  71  55  39  23   7 216
217 201 185 169 153 137 121 120 104  88  72  56  40  24   8
  9 218 202 186 170 154 138 122 106 105  89  73  57  41  25
 26  10 219 203 187 171 155 139 123 107  91  90  74  58  42
 43  27  11 220 204 188 172 156 140 124 108  92  76  75  59
 60  44  28  12 221 205 189 173 157 141 125 109  93  77  61
 62  46  45  29  13 222 206 190 174 158 142 126 110  94  78
 79  63  47  31  30  14 223 207 191 175 159 143 127 111  95
 96  80  64  48  32  16  15 224 208 192 176 160 144 128 112
113  97  81  65  49  33  17   1 225 209 193 177 161 145 129
```

```
10 REM***********************
15 REM***MAGIC SQUARES********
20 REM*****SET-UP*************
30 M=19:DIM A(M,M)
40 PRINT:PRINT"Magic Squares"
50 PRINT:PRINT"How many rows (1 to 19)";
:INPUT S
60 IF S<0 OR S<>INT(S) THEN PRINT"ERROR":GOTO
50
70 IF S>M THEN PRINT"ERROR":GOTO 50
80 IF S/2=INT(S/2) THEN PRINT"ERROR - Odd
Numbers Only":GOTO 50
90 REM**GENERATE SQUARE********
100 X=INT(S/2)+1:Y=S:C=1
110 A(X,Y)=C
120 C=C+1:IF C>S*S THEN GOTO 200
130 X=X+1:IF X>S THEN X=1
140 Y=Y+1:IF Y>S THEN Y=1
150 IF A(X,Y)<>0 THEN X=X-2:Y=Y-1
160 IF Y=0 THEN Y=S
170 IF X=0 THEN X=S
180 IF X=-1 THEN X=S-1
190 GOTO 110
200 REM***PRINT SQUARE*********
210 PRINT:PRINT
220 FOR Y=1 TO S:FOR X=1 TO S
230 A=A(X,Y):GOSUB 380:PRINT" ";A$;" ";
240 NEXT X:PRINT:NEXT Y

250 REM***CHECK ROWS & COLS****
260 F=0
270 FOR Y=1 TO S:T=0
280 FOR X=1 TO S:T=T+A(X,Y):NEXT X
290 IF F=0 THEN U=T:F=1
300 IF T<>U THEN PRINT"ERROR - Row 1 &
Row";Y;" Do Not Match":STOP
310 U=T:NEXT Y
320 FOR X=1 TO S:T=0
330 FOR Y=1 TO S:T=T+A(X,Y):NEXT Y
340 IF T<>U THEN PRINT"ERROR - Row 1 &
Col";X;" Do Not Match":STOP
350 U=T:NEXT X
360 PRINT:PRINT"All rows and cols add to ";T
370 STOP
380 REM*****NUM-STRING CONV*****
390 A$=STR$(A)
400 IF LEN(A$)<3 THEN A$=" "+A$:GOTO 400
410 RETURN
```



Basic Flavours

This program is written in Microsoft BASIC, so it should run unchanged on most popular micros. Spectrum owners must insert LET before all assignment statements. The program asks for the number of rows (and, therefore, columns) in the Magic Square, and checks that this is a positive, integral, odd number. It computes and displays the Magic Square, and then, from line 250 on, checks its own output. If this seems unnecessary, then omit lines 250-360.

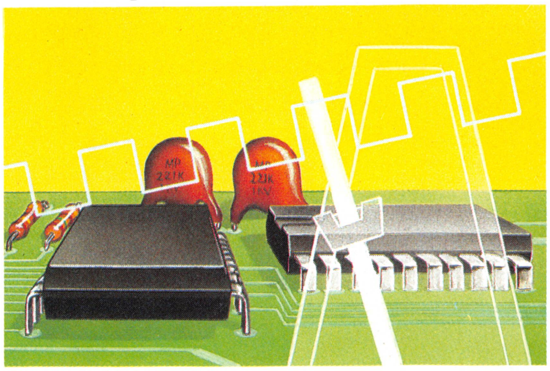
C

CLOCK

The purpose of the *clock* in your home micro, surprisingly, is to slow down the operation of the computer! All the functions of the CPU can be broken down into simple operations performed by logic gates and individual transistors. Though it may seem instantaneous in operation, a transistor takes a finite time to switch its output current in response to an input — although at this level the delay is measured in billionths of a second.

However, these response times will vary from one logic gate to another — even on the same chip — and as the number of logic stages that the data has to pass through increases, so does the possibility of data getting out of synchronisation. A microprocessor, for example, handles eight bits of data in parallel, right through the system. If one of the bits was to arrive at the input to the adder circuitry later than the others, the result produced would be meaningless.

That is why all the digital circuits in a computer, including the memory chips, are synchronised by a central clock. Data cannot progress to the next stage before a clock pulse has been received, and this pulse must therefore be designed to be slower than the response time of the slowest circuit.



The clock itself is simply a quartz crystal, pulsing at a known frequency, with some logic circuitry to divide that frequency down to the one required. Computer clocks give out a few million pulses each second and so are said to have clock speeds of a few *megahertz* (MHz for short). Thus most computers will have clock speeds of 1MHz, 2MHz, 4MHz or even 8MHz.

Companies sometimes try to show how fast their computers are by quoting their clock speeds. This is only useful when comparing computers with the same microprocessor. A computer with a 1MHz 6502 microprocessor is slower than one with a 2MHz 6502, but it can't really be compared with a 2MHz Z80 microprocessor. Even when computers have the same microprocessor, many other things affect their performance, so the clock speed is only a rough guide.

If your machine has a 1MHz clock, that doesn't mean that all operations take one millionth of a second. The clock may well have three *phases*: it produces three independent 1MHz signals, each a third of a cycle apart. On many systems, a memory read is completed in the first phase, and memory write in the second or third phase.

CMOS

A semiconductor material can be impregnated or doped with impurities to make it either *n-type* or *p-type*, according to whether the mobile carriers of electric charge are negative electrons or positive 'holes' (regions where there is a deficit of electrons). The physical arrangement of p-type and n-type areas on a chip determines the circuit or logic gate implemented: a diode is simply an area of one type next to the other, whilst a transistor is a sandwich of one type between two areas of the other type.

Most integrated circuits are based on either PMOS or NMOS technology — the MOS stands for metal oxide semiconductor, and the P or N is indicative of the way in which the types of doping have been arranged to form the circuits.

CMOS means *complementary metal oxide semiconductor*, and is a hybrid of the two other technologies — in simple terms each circuit has two mirror halves that complement each other. CMOS chips consume less electricity than PMOS or NMOS, and are therefore used widely in portable and pocket microcomputers. With a permanent battery connection they can provide non-volatile memory (the contents of which will not be lost when the machine is switched off). For example, the Tandy Model 100 uses CMOS chips; these allow battery operation of the computer itself, and enable data to be stored for many weeks. The main disadvantage of CMOS chips is that they are relatively expensive, although, with increasing popularity, prices are falling fast.

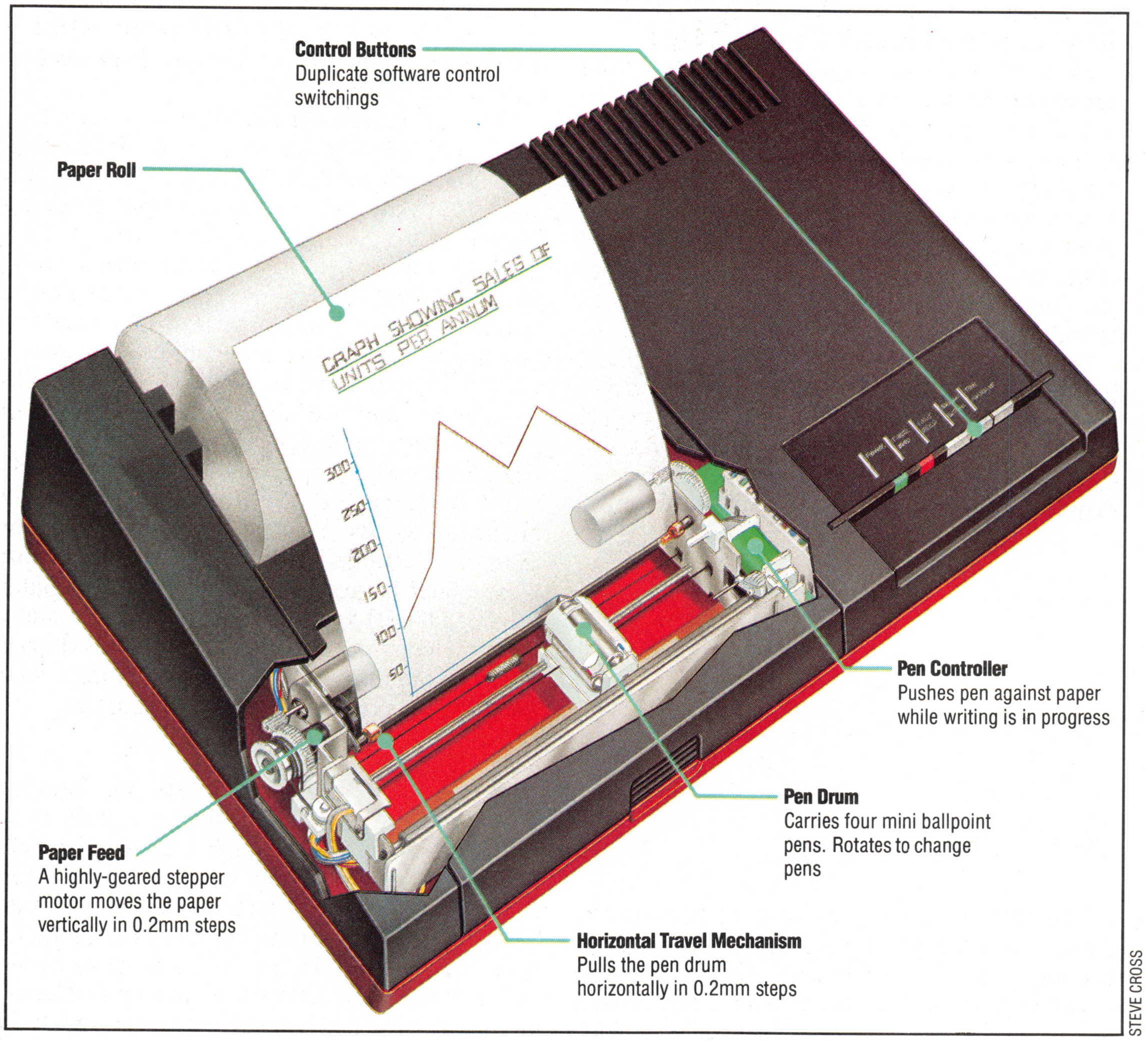
COAXIAL CABLE

Ordinary wire and ribbon cables are fine for transmitting low-frequency digital signals over short distances, but for high-frequency signals they are useless. There are two main reasons for this: the signal is vulnerable to electrical noise from the surrounding environment, and it is prone to start 'seeping' out of the cable. *Coaxial cable*, which consists of a central wire surrounded by an insulation tube and an earthed metal screen, is designed for high-frequency use, and is invariably used for the UHF connection between your micro and television set. Its other main use in computing is to form the link on certain types of local area network (in particular, the Ethernet), in which all the individual stations simply couple onto a common piece of coaxial cable.

COBOL

The COmmon Business Oriented Language was the first programming language written for commercial applications. Languages developed before this were designed for scientific and mathematical programming. Though much commercial software was written in COBOL for mainframes and minicomputers, little use has been made of it for micros. Nevertheless, COBOL is available for machines running CP/M — the most widely acclaimed version being CIS COBOL produced by Microfocus.

PEN PALS



Control Buttons
Duplicate software control switchings
Paper Roll
GRAPH SHOWING SALES OF UNITS PER ANNUM
300
250
200
150
100
50
Pen Controller
Pushes pen against paper while writing is in progress
Pen Drum
Carries four mini ballpoint pens. Rotates to change pens
Paper Feed
A highly-geared stepper motor moves the paper vertically in 0.2mm steps
Horizontal Travel Mechanism
Pulls the pen drum horizontally in 0.2mm steps
STEVE CROSS

Family Plot
The printer/plotter has a resolution of roughly 500×2000 pixels in plotter mode, and a choice of type sizes. The pen colour is selected by rotating the pen drum, and plotting or printing involves moving the pen drum horizontally left and right and scrolling the paper up and down. The pen can be pushed against or pulled away from the paper, to permit plotting or trace-free movement respectively. The mechanism is capable of high-accuracy plotting, but repeated movement of paper and drum causes a slight loss of registration

Plotters are complex devices, which are used for producing detailed colour graphics on paper. As such, they are priced beyond the means of most home computer users. However, there is one model — costing just over £100 — that appears under a variety of different names and is eminently suited to use with micros.

The Atari 1020, Commodore 1510 and Oric MCP-40 are just some of the names under which a remarkably cheap microcomputer plotter is sold (a marketing technique called *badge engineering*). The device behind all these machines is made by a Japanese manufacturer and adapted to each company's requirements.

In addition to its graphics capability, the plotter will also produce text, and is thus referred to as a *printer/plotter*. Inside the unit is a revolving head, which contains four small ballpoint pens. To draw a line, the head is rotated to select the correct colour (red, blue, green and black are fitted as standard), and the chosen pen is then moved into contact with the paper. A horizontal line is drawn as the head moves from side to side; a vertical line is produced by up and down movement of the paper. Text is produced in much the same way as graphics — the printer/plotter stores the patterns for letters and other characters in its own memory. When a signal is received from the computer, the printer/plotter simply finds the relevant character in its internal memory and then draws it as if it were a graphics pattern. The resulting print quality is extremely good — certainly better than the majority of cheap dot matrix printers.

When in text mode, the printer/plotter operates in exactly the same way as any other printer. Although the paper used is only 115 mm (4.5 inches) wide, the unit will produce either 40 or 80 characters per line. The narrow paper width does mean that 80-character text is somewhat small, but character definition is extremely good. Text may be printed in any of the four colours, with a printing speed of around 10 characters per second. This is slow in comparison to most dot matrix printers, but roughly equal to the speed of daisy wheel output.

To produce graphics, the printer/plotter is switched into graphics mode and will then produce lines, curves and large letters. If the unit is sent characters while in graphics mode it will interpret them as commands and will not attempt to print them. Thus, if line-drawing is required, a command such as:

```
LPRINT "D 0,100,100,100,100,0,0,0"
```

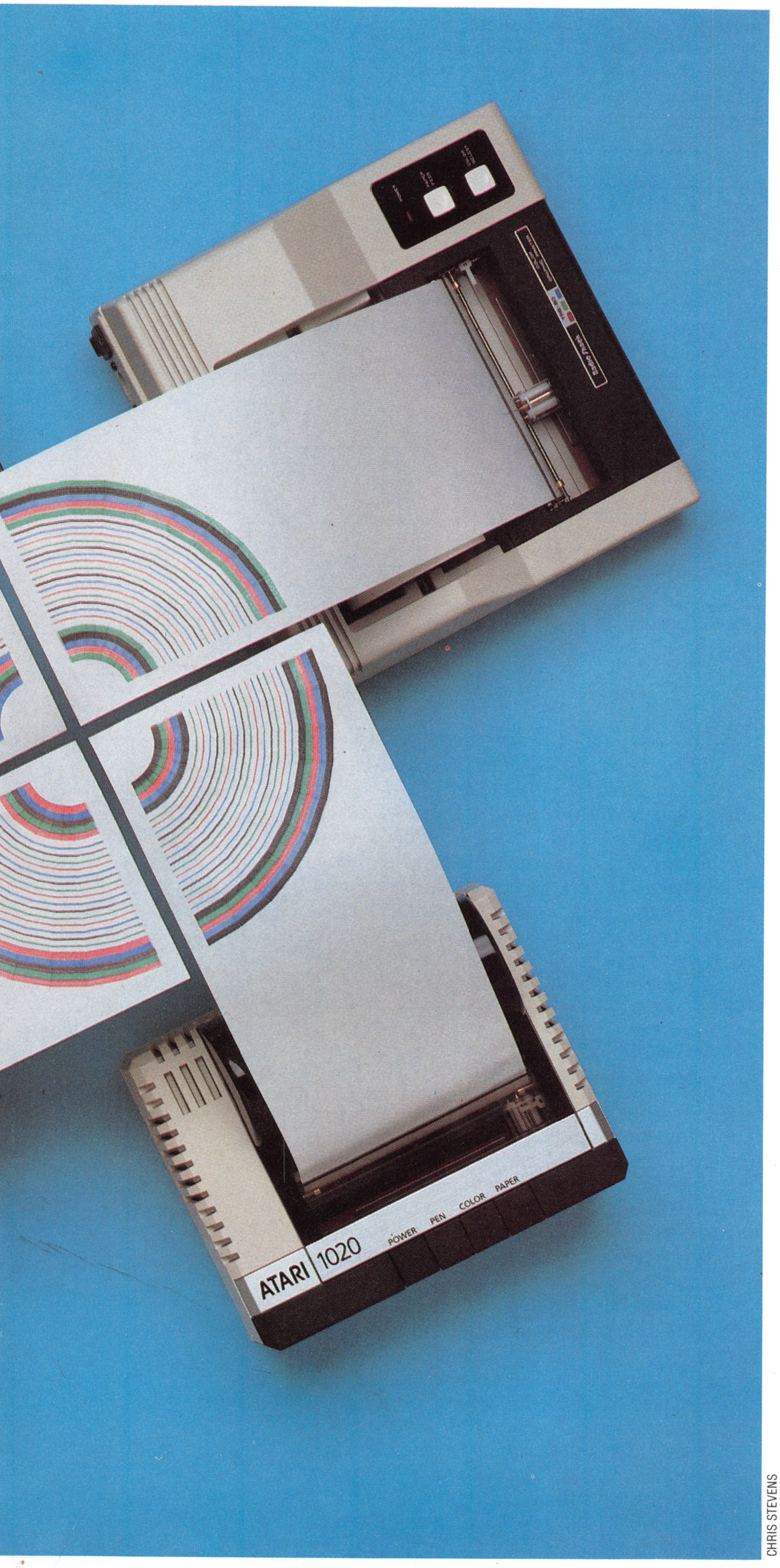
would be used in BASIC. The D is the instruction to draw lines and the numbers following give the positions of the points through which the lines will pass. In this case (assuming that the print head is positioned at the co-ordinates (0,0) to begin with) a square will be drawn. Other commands follow a similar format.

The width of the paper is regarded as being divided into 480 horizontal steps, which are used to position the print head. The paper used is supplied on a roll that is several metres in length. This might give the impression that there is no limit to the vertical length of a drawing, but in fact paper slippage can result in lines failing to join up as required, so the printer/plotter will not allow the paper to be wound back more than a certain distance.

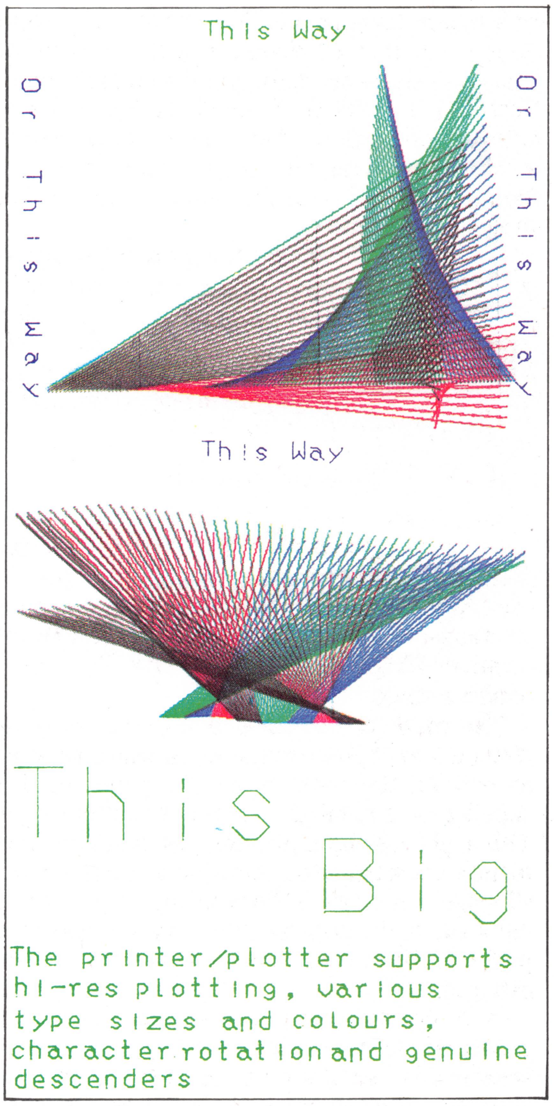
Although some very complex graphics may be produced on the printer/plotter, programs to achieve this take a long time to write — there is no easy way of copying an image directly from the computer screen. Line drawing is simple enough, but the absence of a PAINT or FILL command means that blocks of solid colour are obtainable only by drawing many fine lines. This is slow and wasteful of ink. The device is also unable to draw to the full paper width — a small gap must be left at each side. The unit is especially suited to graph-drawing, and features a command that automatically draws the axes of a graph, complete with the relevant units marked at chosen intervals.

In text mode, the printer/plotter will produce two different print sizes. However, by using graphics mode, much larger text sizes are available and letters may be printed sideways, allowing large messages to be printed along the length of the paper. The ballpoint pens are quick to run out of ink, and are likely to dry out if left in the print head. Each time the unit is turned on, all the pens are tested automatically as the printer/plotter draws a small square in each colour.

Professional plotters are extremely accurate, and produce high-quality colour graphics. It would be unreasonable to expect a unit in this price range to equal these standards, but it must be said that the results achieved with the printer/plotter are surprisingly good.



CHRIS STEVENS



The printer/plotter supports hi-res plotting, various type sizes and colours, character rotation and genuine descenders

Plotter Interfaces
The chart shows which units can be used with the popular micros. The only problem will be getting the right lead to connect them together. Oric, Sharp, Commodore and Atari include leads suitable for their own micros.

One notable machine missing from the list is the Sinclair Spectrum, which lacks a suitable interface.

	Atari 1020 £200	Commodore 1520 £170	MCP-40 £148	Oric MCP-40 £150	Sharp MZ-80P5(K) £130	Tandy CGP-115 £129
Atari 400/600/800	●					
BBC Micro			●	●		●
CGL/Sord M5			●	●		●
Colour Genie						●
Commodore Vic-20		●				
Commodore 64		●				
Dragon 32/64			●	●		●
Memotech MTX			●	●		●
Oric-1/Atmos			●	●		●
Sharp MZ-700					●	
Tandy TRS-80 Colour						●

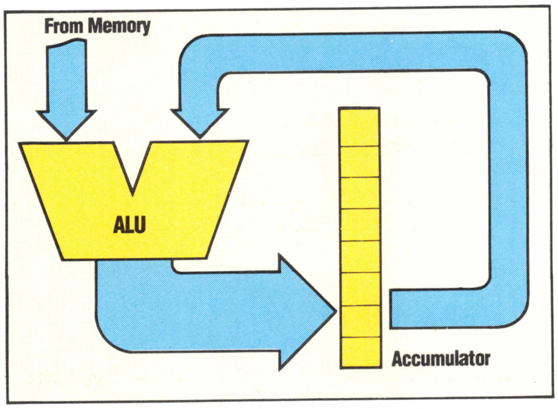
LOGICAL END

The central processing unit (CPU) controls the transfer of data around the computer, oversees the execution of program instructions, and carries out arithmetic and logical calculations. In this final instalment of the course, we look at the logic of the CPU and, in particular, the function of the arithmetic and logic unit (ALU).

The functions that the ALU performs fall into two categories: arithmetical functions — addition, subtraction and incrementation (adding one to a number) — and three logical functions — Exclusive OR, Inclusive OR and AND.

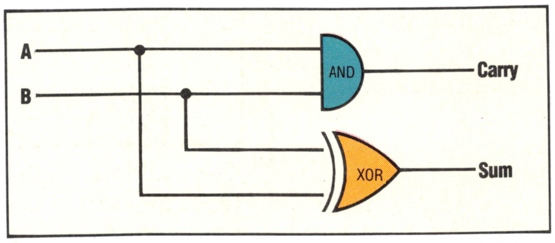
Some of these functions, such as addition, require two operands (i.e. numbers on which the operation is carried out). Some, such as incrementation, require only one operand. In the latter case, the required operand is taken from a special register within the CPU called the accumulator. Where two operands are required, the second is fetched from main memory. The two numbers then progress through the circuitry of the ALU and the selected operation is carried out. The result of the operation is finally placed back in the accumulator.

The following diagram shows the flow of data through the ALU chip:

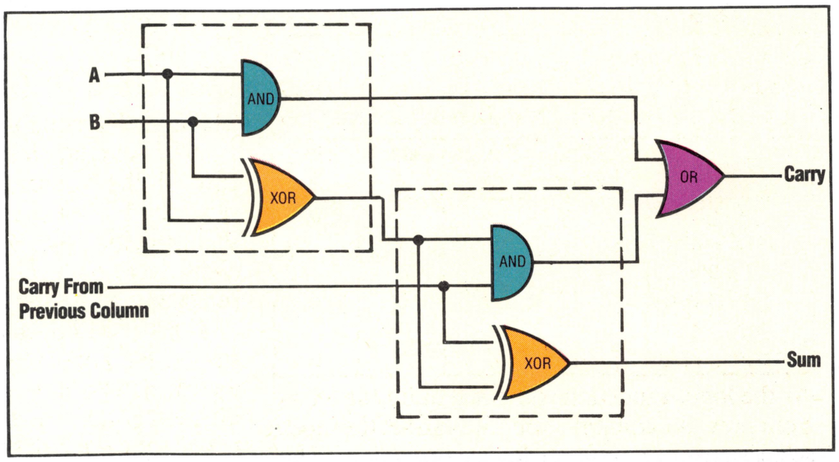


The numbers in the accumulator and memory have an eight-bit word length. The bits that make up the numbers travel through the ALU in parallel and the operation is carried out on all eight bits simultaneously. In order to describe the circuitry of the ALU, let us take just one of the eight bits and design a circuit that will allow us to carry out all six of the different functions. We shall call the bit from the first operand A and that from the second operand B.

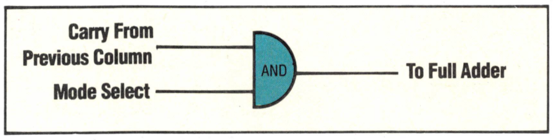
The basis for the one bit stage of the ALU we are considering is the full adder. The full adder circuit we designed in a previous instalment (see page 46) used two half adder circuits, which consisted of AND, OR and NOT gates. These half adders, however, can make use of an Exclusive OR gate to simplify the circuitry:



The two half adders link together to form the full adder circuit in this way:

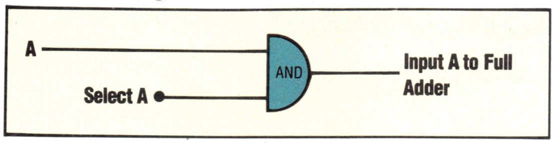


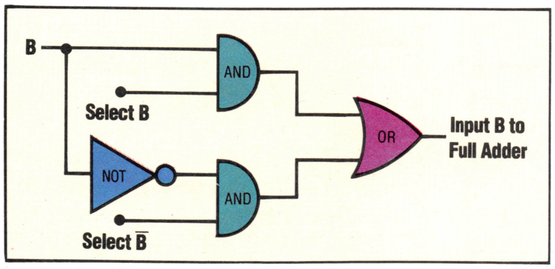
By adding on small circuits to the full adder it is possible to show how the adder's gates can be adapted to perform the other functions of the ALU. We shall set up a series of control signals to do this, the first of which is known as the *mode select* signal. The 'carry from previous column' input is used during arithmetic operations but is not required during logical operations. The mode select signal switches the carry input on or off. This is easily achieved by using an AND gate.



For arithmetic operations where the carry input is required, the mode select signal should be set to one, allowing the carry input to pass through the AND gate. When the carry input is not required then the mode select signal should be set to zero. In a similar way, we can add AND gates to the two other inputs. This allows us to select either bit A, bit B or both bits together.

During the process of subtraction by two's complement additions (see page 278), the two's complement of the number to be subtracted must be calculated. This requires the changing of all ones to zeros, and vice versa. If our adder circuit is to be used for subtraction then we must add some circuitry that will allow us to select the negation of bit B. This can be done by feeding the input B through a NOT gate, and the select signal can be incorporated by means of another AND gate. The final circuitry for the inputs A and B together with the control signal is as follows:



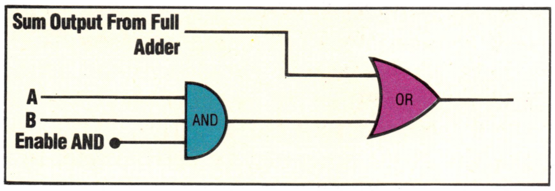


Using these four control signals, each of the arithmetic functions can now be accomplished. This table shows the required combinations:

Function	Mode Select	Select A	Select B	Select B̄
Addition	1	1	1	0
Subtraction	1	1	0	1
Increment A (Set 1st carry input to 1)	1	1	0	0
Increment B	1	0	1	0

As the logical functions do not require the 'carry from previous column' input, we can set the mode select signal to zero for all logical operations. This means that the logical XOR function can be achieved at the sum output by setting Select A and Select B HI, and setting the mode select signal LO.

The AND function cannot be taken directly from the existing circuitry and requires separate A and B inputs, together with an AND select signal.



Finally, the OR function can be created by combining the XOR and AND outputs through an OR gate. The following truth table demonstrates this:

A	B	Output	Comments
0	0	0	
0	1	1	XOR Function
1	0	1	
1	1	1	AND Function

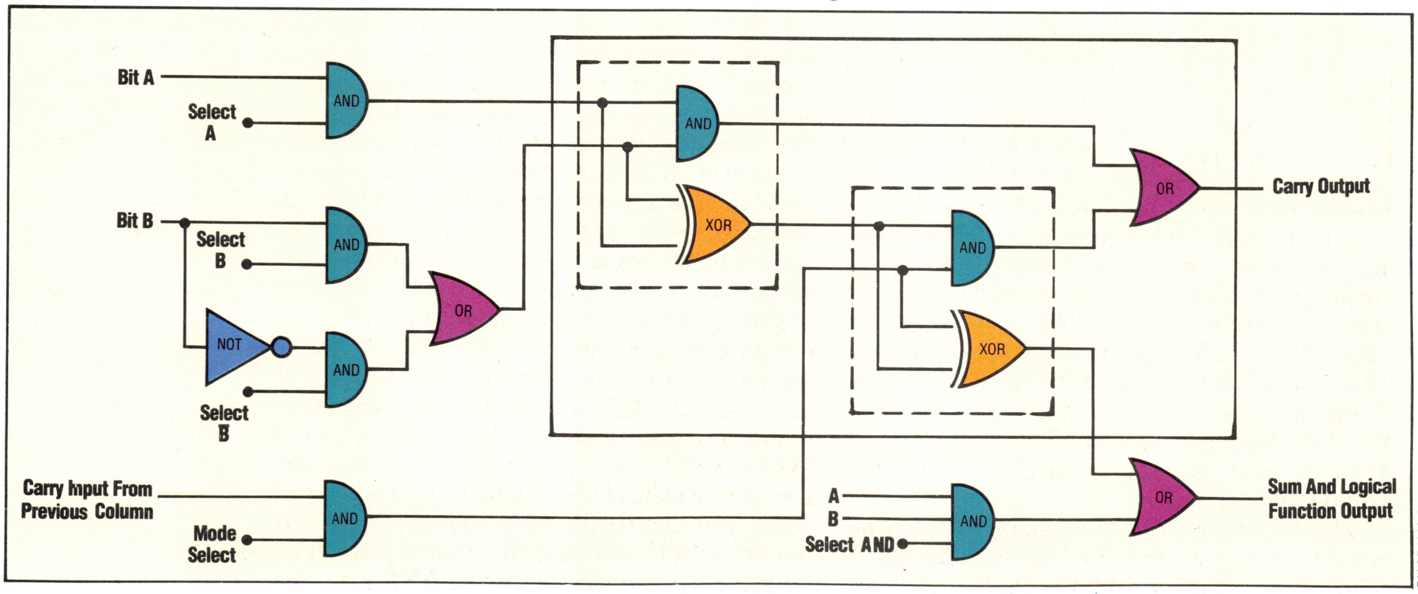
The following table shows how the logical functions can be produced using different combinations of the control signals:

Function	Mode Select	Select A	Select B	Select B̄	Select AND
XOR	0	1	1	0	0
AND	0	0	0	0	1
OR	0	1	1	0	1

The final diagram below shows a one-bit stage of an ALU circuit, illustrating the full adder circuit and all the additional circuits for the control signals. The full circuit would incorporate eight such circuits in parallel. The carry output from the eighth column is used as the carry flag for the processor status register.

This article concludes our series on logic. We started the course by dealing with such abstract logical concepts as Boolean algebra and Venn diagrams, and then considered simple logic circuits and the results they give. The last articles in the series investigated more complex circuits at a level close to that used inside a computer.

Finally, we have shown how it is possible to combine several circuits in a way that allows a microprocessor to perform all the arithmetic and logical operations it needs to do. Each operation can be performed by putting the right patterns of ones and zeros on the command lines of the ALU. These patterns are in fact machine code instructions in their true binary form. Thus we have studied the logic of the hardware inside computers up to the point where software takes over as the controlling factor.



DIAGRAMS BY LIZ DIXON

FOR THE RECORD

In this series on file handling we've looked at the basic principles behind all computer filing systems. However, it is important to realise that file handling methods tend to be machine-specific. In this final instalment, therefore, we examine the ways in which these techniques may be used on cassette-based micros.

Different micros use different file handling techniques, and therefore it is often necessary to adapt a standard program to run on a particular machine. For this reason, it is important to understand how the facilities and commands offered by your own system relate to the general methods we have discussed. As an example, let us examine the storage of data files on a standard cassette-based micro. The first point to notice is that cassette systems, by their very nature, cannot handle random access files (see page 244), so data must be accessed in the order in which it is stored — that is, sequentially.

As sequential file handling involves reading information from one file, working on this information and then writing the modified data into a second file, it is obvious that two files must be in use ('open') at the same time. A cassette recorder is incapable of moving directly and accurately between two tape positions, so this 'two-file' system is not suitable for most cassette-based micros. The exceptions to this rule are those few micros, such as the Newbrain and Commodore PET range, that provide two cassette ports — one for reading, one for writing.

Most home machines, therefore, are limited to one sequential file at a time. In practice, this imposes some major limitations. The file must be read into memory before it is used, and then, if changes have to made, it must be written out onto cassette again. This may be done at intervals during program operation, or once at the end of the program run. Data files must be small enough to reside in RAM after space has been taken up by the filing program itself. Most home micros are thus restricted to small data files.

Three main methods have evolved for storing information on cassette. The simplest system does not use separate data files at all; instead all current variables are stored along with the program whenever the SAVE command is used. This method is used on the ZX81 and is also available on the Sinclair Spectrum. When a new data file is required, a 'fresh' copy of the program is used and then SAVEd along with its data. When this version is next LOADed, the data is automatically read back into the required variables. The virtue of this method is its simplicity — all the user has to do is to make sure that the complete program is correctly SAVEd and LOADed.

A slightly more sophisticated system requires a BASIC that is able to store and read back specific arrays. On the Oric Atmos, for example, the command STORE A$, "NAME" will write the array A$ to tape, and RECALL A$, "NAME" will read it back again. The whole array (A$(1), A$(2), etc.) is SAVEd, although the STORE and RECALL commands don't actually specify the array size, which is given automatically when the array is DIMensioned at the start of the program.

One problem with this system is keeping track of the number of entries in the array that are used in your program. One solution is to store the record count in the array before it is SAVEd. Most machines allow a zero subscript, so an element such as A$(0,0) can be used for the record count. The record count will be a numeric variable (in our program we use R), but if a string array is being used this must be converted into a string. This is done quite simply with a line such as: A$(0,0) = STR$(R). Once the array has been reloaded into the computer, R is reset with: R = VAL(A$(0,0)).

Although many micros do not support sophisticated file handling procedures, these may be simulated, as the listings here show. Once the file is safely installed in memory, it is easy to use BASIC arrays to treat it as a random access file.

Let's assume that a two-dimensional string array is used to store the data; this may be set up with a command such as DIM A$(100,3). The first subscript in the array may be used to refer to a particular record, and the second subscript will point to one of four fields. This allows the data to be stored in the familiar table format and is equivalent in operation to a random access file.

Another useful facility present on some machines is the APPEND command. This allows you to add data to the end of the sequential file without first reading through it and then creating a new version. Some computers have a command that allows a number of fields in a sequential file to be skipped over and this provides a simple random access facility.

This series has concentrated on the fundamental aspects of a complex topic. File handling is very much machine-dependent, and all the ideas that we have presented in these articles will need to be adapted for your own machine. But the basic principles will be the same, no matter what micro you use, and you should find much of the material useful when writing your own programs.

Storing Records And Fields In A Basic Array

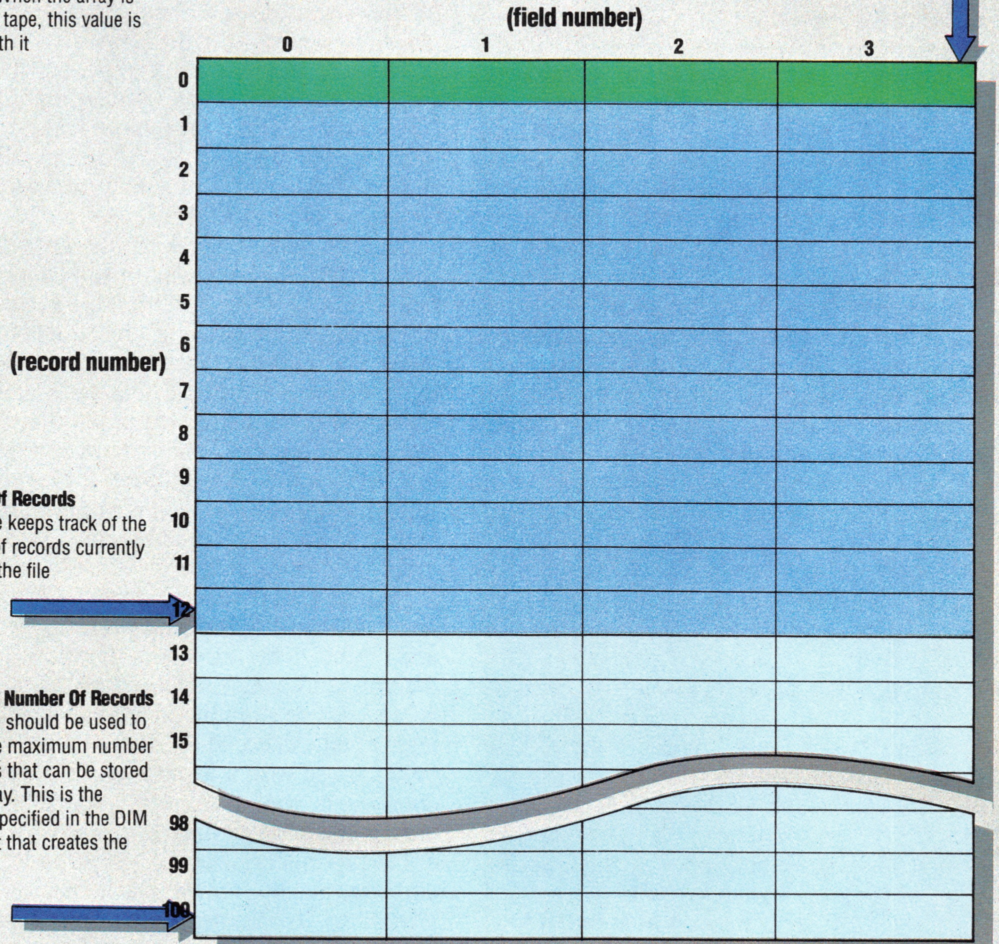
A two-dimensional string array can be used to simulate a random access file. The first subscript is used to identify a particular record and the second identifies fields within a record

Number Of Fields
It's handy to have this number defined in a variable at the start of the program. This makes it easier to change to using larger records later, because it would mean altering only one instruction

Header Record
A$(0,0) is used to store a count of the number of records. When the array is SAVEd to tape, this value is stored with it

Number Of Records
A variable keeps track of the number of records currently in use in the file

Maximum Number Of Records
A variable should be used to record the maximum number of records that can be stored in the array. This is the number specified in the DIM statement that creates the array



Deleting A Record From The Array
This program segment removes record number N from the array using the method we have already detailed. All the records below N are moved up one space, overwriting the data currently stored in position N

```
100  LET R = R + 1
110  IF R > M THEN  PRINT "ARRAY FULL": RETURN
120  FOR I = R TO N + 1 STEP  - 1
130  FOR J = 0 TO F
140  LET A$(I,J) = A$(I - 1,J)
150  NEXT J: NEXT I
170  LET A$(N,0) = N$: LET A$(N,1) = C$
180  LET A$(N,2) = D$: LET A$(N,3) = E$
190  RETURN
```

Inserting A Record Into The Array
This program segment inserts a new record into the array at position N in the file. All records past the point of insertion are moved down to create a gap for the new record stored in N$, C$, D$ and E$

```
200  FOR I = N TO R - 1
210  FOR J = 0 TO F
220  LET A$(I,J) = A$(I + 1,J)
230  NEXT J: NEXT I
240  LET R = R - 1
250  RETURN
```

KEVIN JONES

Loading And Saving A Record Array

Variables SAVEd with Program
The Spectrum saves all its variables along with any saved programs (although it can also save arrays on their own). The sample program fills an array with data and the SAVE instruction will store the array on tape with the program. When the program is loaded again it will automatically start at line 700 because the SAVE instruction refers it to LINE 700. The RUN instruction must not be used because it clears all variables

Spectrum

```
100 REM***SPECTRUM DEMO***
200 DIM A$(100,20)
300 FOR K=1 TO 100
400 LET A$(K)="RecNo."+STR$(K)
500 NEXT K
600 STOP
700 PRINT "ARRAY CONTAINS..."
800 FOR K=1 TO 300
900 PRINT A$(K),
1000 NEXT K
1100 STOP

SAVE "DEMOPROG" LINE 700

LOAD "DEMOPROG"
```

SAVEing Named Arrays
The Oric Atmos provides the commands STORE and RECALL to save and load particular arrays from tape. This makes it easy to store a file held in an array

Oric Atmos

```
100 REM Save array to tape
110 A$(0,0)=STR$(R)
120 PRINT "Please position tape,
press PLAY and  RECORD then hit RETURN"
130 A$=KEY$:IF A$="" THEN 130
140 STORE A$,"AFILE",S
150 PRINT "FINISHED"
160 RETURN

200 REM Load array from tape
210 PRINT "Please position tape
 and press PLAY then   RETURN"
220 A$=KEY$:IF A$="" THEN 220
230 RECALL A$,"AFILE",S
240 R=VAL(A$(0,0))
250 PRINT "FINISHED"
260 RETURN
```

Using Serial Files
The BBC Micro is one of several home computers to support true serial files on cassettes. These two subroutines store and reload the array by creating a serial file. The first field is the record count

BBC Micro

```
100 REM Save array to tape
105 PRINT "Please position tape"
110 X=OPENOUT("AFILE")
120 PRINT£X,R
130 FOR I= 1 TO R
140 FOR J= 1 TO F
150 PRINT£X,A$(I,J)
160 NEXT J: NEXT I
170 CLOSE£X
180 PRINT "FINISHED"
190 RETURN

200 REM Load array from tape
205 PRINT "Please position tape
and press PLAY then RETURN"
210 IF INKEY(0)=-1 THEN 210
215 X=OPENIN("AFILE")
220 INPUT£X,R
230 FOR I= 1 TO R
240 FOR J= 1 TO F
250 INPUT£X,A$(I,J)
260 NEXT J: NEXT I
270 CLOSE£X
280 PRINT "FINISHED"
290 RETURN
```

JUMPERS

Bugaboo (also known as Booga-boo) is an unusual computer game for the Spectrum and Commodore 64 in which the player takes the role of a flea that gets trapped in a deep pit. For a top-selling program, the game is surprisingly simple — but the sophisticated graphics and fast action make it a delight to play.

The impressive title sequence of Quicksilva's Bugaboo game informs you that you are approaching the planet Cebella-7, on which there is evidence of life. After landing safely, you hop around in the lurid flora on the planet surface before losing your footing and tumbling into a deep pit, which opens out into a large cave. The object of the game is to escape from the abyss.

Ledges protruding from the gaudily-coloured rocks provide footholds in a landscape reminiscent of Hieronymus Bosch — multi-coloured mushrooms and flowers abound, and spiders cling to the walls of the cavern. You must attempt to escape by jumping from ledge to ledge, all the while avoiding a flea-eating dragon. In the Commodore version only, Venus flytraps provide a further floral hazard.

The screen acts as a window on the playing area in both versions, and new sections of the cavern are revealed as you move around and climb higher. The cavern is approximately three screens wide and five screens high, so it takes time to discover all the hazards. You may take any one of several different routes, and the ledges are cunningly placed to increase the difficulty.

The scenario may be original, but the game is spoilt to a certain extent by the program coding. You have one life only, and the introductory instruction sequence is repeated each time this is lost — which happens a lot, as the dragon proves to be almost impossible to avoid.

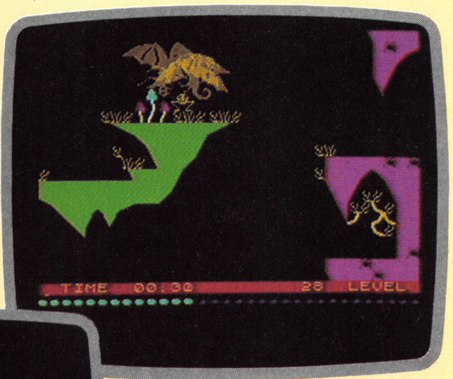
As the game progresses, the ledges become harder to reach. But no hazards are introduced at higher levels, which is unfortunate as the program would be considerably more addictive if other predators appeared in the later stages. Storing multiple screens uses a lot of memory, leaving little for complex rules and action, but there are many other programs that manage to pack more in — Jet Set Willy is a good example. The Commodore 64 version features more complex graphics than the Spectrum version, but otherwise fails to make the most of its extra memory.

The Spectrum version is keyboard-controlled, with the '1' and '0' keys triggering jumps to the left and right. The strength of these jumps is determined by the length of time the relevant key is held down: the longer the key is pressed, the higher you will go. The Commodore 64 version is only for use with a joystick, with the fire button controlling the scan — a more satisfactory arrangement if you have a good joystick.

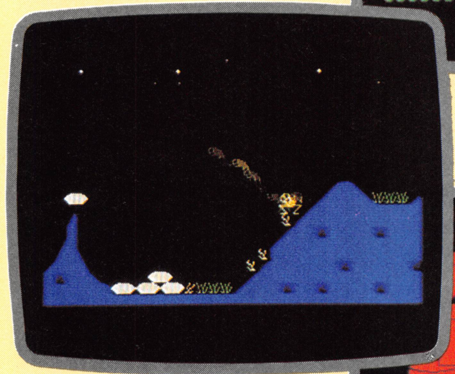
Despite the criticisms of Bugaboo's long-term addictive quality, it must be said that the game has plenty to recommend it. The controls are extremely simple, allowing the user to start playing immediately, without having to keep returning to the instruction screen. And the imaginative graphics are simply stunning.

Jump To It

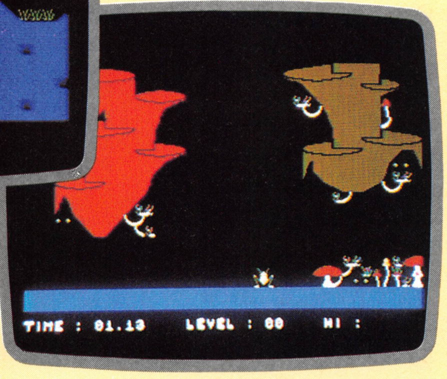
The outstanding feature of Bugaboo is its detailed graphics. These depict a deep pit consisting of many ledges that the player (impersonating a flea) has to jump between in order to ascend to freedom. The Commodore's graphics have a slight edge over those in the Spectrum version



Bugaboo (The Flea) On The Spectrum



Bugaboo (The Flea) On The Spectrum



Booga-boo (The Flea) On The Commodore 64

SCREEN SHOTS BY IAN McKINNELL

Bugaboo (The Flea): for 48K Spectrum, £6.95

Booga-boo (The Flea): for Commodore 64 + joystick, £7.95

Publishers: Quicksilva, P.O. Box 6, Wimborne, Dorset BH21 7PY (0202 891744)

Authors: Indescomp & Microbyte

Joysticks: Commodore version only

Format: Cassettes

MULTIPLE CHOICE

As we continue our investigation of Assembly language arithmetic, we consider the problems associated with subtraction, and the various ways of dealing with them. We also begin to look at the programming of multiplication in machine code, and introduce a new class of logical operations — the Shift and Rotate op-codes.

Both the Z80 and 6502 support the SBC (SuBtract with Carry) instruction, but their implementations are quite different. On the 6502, the carry flag is used to handle the *borrow* facility, which is the equivalent in subtraction of the carry facility in addition. In Z80 Assembly language, SBC works in exactly the same way as the ADC instruction — the carry flag is set or reset to indicate the result of the operation.

Suppose that we add $E4 to $5F using ADC (having cleared the carry flag first). The result in the accumulator is $43, and the carry flag is set, showing that the true result is $0143. There has been an overflow into the carry flag because the accumulator cannot contain the full result.

Now suppose that on the Z80 we again clear the carry flag, and subtract $E4 from $5F: the result in the accumulator is $7B, and the carry flag is set. If we now add $7B to $E4 (having cleared the carry flag once again) we find the result in the accumulator to be $5F, and the carry flag is set. This is entirely consistent, as can be seen:

```
$5F - $E4 = $7B        Carry Set
$5F = $E4 + $7B        Carry Set
```

If we take the carry flag's state as indicating that a negative result has occurred, then we can interpret $7B as a two's complement number:

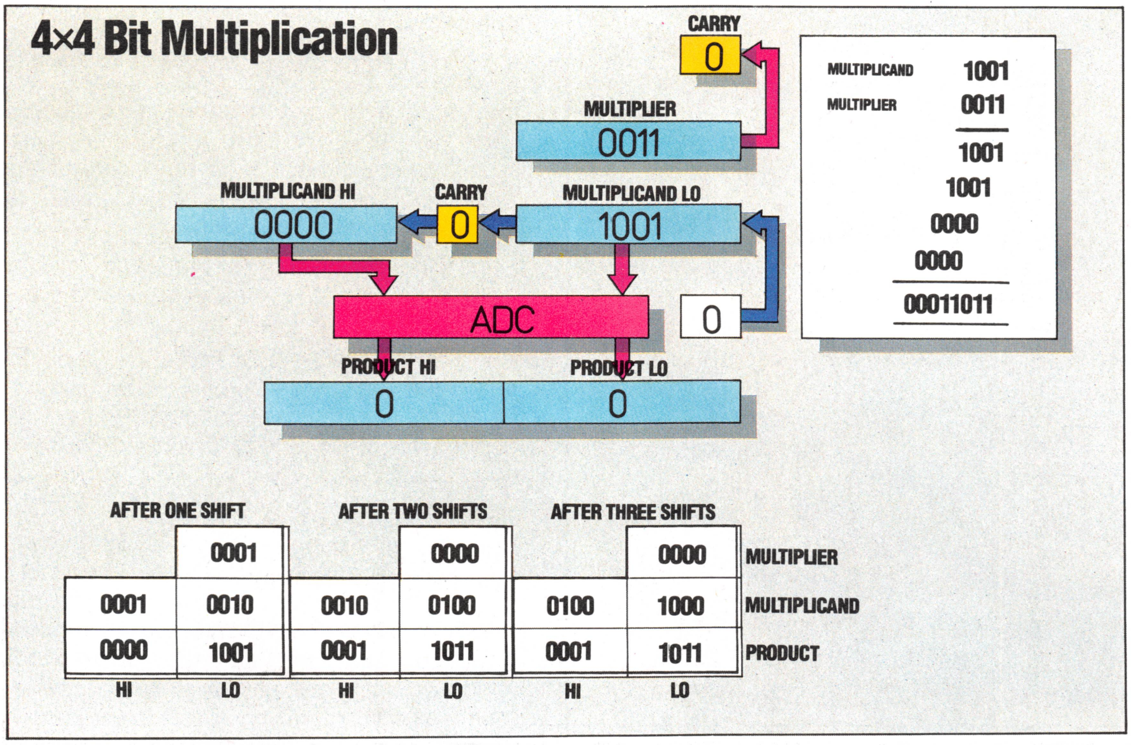
```
$7B In Binary                    = 01111011
Take Away One                           - 1
                                 ----------
Gives One's Complement             01111010
                          Negate
Gives Two's Complement             10000101 = $85
```

We should expect to find, then, that $5F — $E4 results in the negative number –$85. Let's check this result in decimal:

```
  $5F =     95 decimal
 -$E4 =    228 decimal
 ------------------------
  $85 =   -133 decimal
```

Clearly, this all makes sense as far as it goes. Suppose now that the subtraction in question was actually a two-byte sum: $375F — $21E4.

```
  HI    LO
  $37   5F        =   14175 decimal
 -$21   E4        =  -8676 decimal
 ------------------------------------
  $15   7B        =   5499 decimal
```



	AFTER ONE SHIFT HI	AFTER ONE SHIFT LO	AFTER TWO SHIFTS HI	AFTER TWO SHIFTS LO	AFTER THREE SHIFTS HI	AFTER THREE SHIFTS LO
MULTIPLIER		0001		0000		0000
MULTIPLICAND	0001	0010	0010	0100	0100	1000
PRODUCT	0000	1001	0001	1011	0001	1011

Shift Times
This example shows four-bit multiplication for the sake of clarity—the number of bits does not affect the algorithm. The worked example shows how the product is formed by the addition of zeros or shifted versions of the multiplicand, depending on whether each bit of the multiplier is zero or one. The multiplier bits are right-shifted through the carry flag, while the multiplicand bits are left-shifted from lo-byte to hi-byte through the carry flag

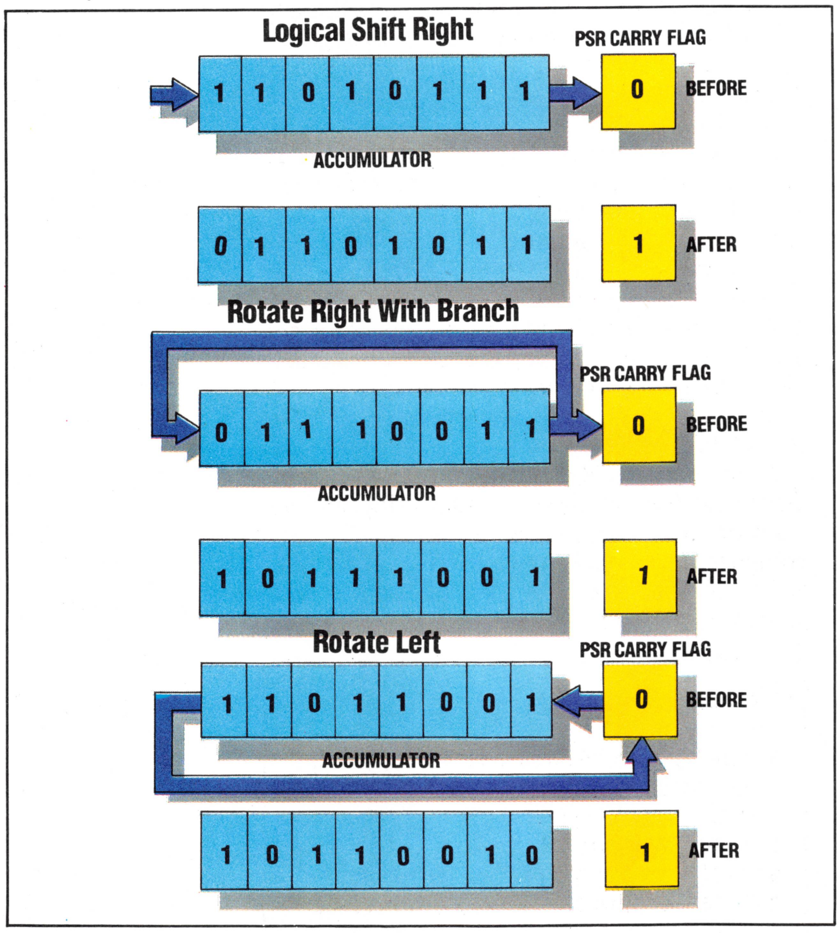
We know that if we do the lo-byte subtraction first the result is $7B, and a carry. That carry is then added by the SBC instruction to $21, making $22, which is then subtracted from $37, giving $15. The answer, $157B, can be seen to be correct by checking the decimal version.

Two-byte arithmetic on the Z80, therefore, follows this simple procedure:

1) Clear the carry flag.
2) Subtract the lo-bytes with carry.
3) Subtract the hi-bytes with carry.

The 6502 version of this sequence differs in the first particular — the carry flag must be set to permit a 'borrow' out of the lo-bytes from the hi-byte. If no borrow occurs, then the subtraction proceeds as normal, and the carry flag remains set for the subtraction of the hi-bytes, which should similarly proceed normally. If an underflow occurs in the lo-byte subtraction, however, the carry flag acts as the 'ninth bit' of the accumulator. This ensures that a correct result occurs there, and that the carry flag is then reset. When the hi-bytes are subtracted with a reset carry flag, the effect is the same as in the Z80 hi-byte subtraction with the carry flag set — the number to be subtracted is decremented before the subtraction takes place. Both methods of dealing with the subtraction borrow have their equivalent in the old-fashioned arithmetic methods of 'borrowing' here, and 'paying back' there. Let's consider the 6502 version in more detail.

Shift Work
The Shift and Rotate instructions are used primarily to examine the contents of a register bit by bit. With each shift, the top or bottom bit of the register is moved into the PSR carry flag; the state of the carry flag can then be used by a branch instruction to determine the flow of program control (as can be seen in the multiplication subroutine in this instalment). The rotate instructions can be used so that register contents are preserved, but the logical shift instructions shift zeros in as they shift bits out. A left shift, therefore, multiplies the register contents by two, and a right shift divides the contents by two



If we clear the carry flag, and subtract $E4 from $5F, the result is $7A in the accumulator, and the carry flag remains clear. We have seen from the Z80 example that a 'true' result is $7B with the carry flag indicating a negative number. $7B is the two's complement of the 'real' answer (–$85). We can see that $7A is the one's complement of this number, and that the state of the carry flag, therefore, is a kind of switch on the accumulator's mode. That is to say, it is set for two's complement, and reset for one's complement.

If we do the subtraction on the 6502 with the carry flag set, then the accumulator contains $7B, and the carry flag is reset. If this is a two-byte subtraction, putting the carry flag into reset state will ensure that the hi-byte subtraction result is decremented, thus taking care of the 'borrow' from the lo-bytes.

MULTIPLICATION

Consider the decimal multiplication sum:

```
   174      Multiplicand
  ×209      Multiplier
  ----
  1566      1st Partial Product
  000       2nd Partial Product
+348        3rd Partial product
  -----
 36366      Final Product
```

You don't have to understand positional notation to use this method, you just have to be able to follow simple procedures and do single-digit multiplication. The heart of the method is the writing of each partial product one place to the left of the previous product (the empty columns are left blank here for emphasis). Once the necessity for this is accepted, then forming the partial products requires only a knowledge of the multiplication tables.

The combination of shifting partial products and rote learning of tables is what makes decimal long multiplication difficult for many people. There is only one real product in binary multiplication, and that is one times one; all other single-digit products result in zero. Consider this binary long multiplication sum:

```
   1101    =  13 decimal
   1001    =  9 decimal
   -------
   1101       1st Partial Product
  0000        2nd Partial Product
 0000         3rd Partial Product
1101          4th Partial Product
-------
1110101    =  117 decimal
```

The shifting of partial products is clearly seen in this example, as is the overall simplicity of multiplication in binary. A partial product is equal to either zero or to the shifted multiplicand, depending on whether the corresponding multiplier bit is one or zero. That immediately sounds like the sort of test we've become used to as an Assembly language control structure. To perform binary multiplication, examine each bit of the multiplier in turn, and add zero (if the bit is zero) or the shifted multiplicand (if the bit is one) to the total. How, then, do we examine a single bit of the multiplier, and how do we shift the multiplicand?

Testing the state of a particular bit in a byte can be done using the BIT instruction on both the Z80 and 6502 microprocessors. On the Z80, this instruction takes an address and a bit number as its operands, and the zero flag is set if that particular bit is zero and reset if the bit is one. On the 6502, the operand is an address. The contents of this address are ANDed with the accumulator, and the zero flag is set or reset, depending on whether the result is false or true.

These instructions permit subtle programming, but neither method is particularly convenient here. It would be much more convenient if the bit in question could be made to act as the carry or zero flag, so that program flow would branch automatically according to the state of each bit in turn. Needless to say, both processors' instruction sets make that possible through the use of the *shift* instructions. As the name implies, these will also solve the problem of shifting the multiplicand.

There are a variety of shift and *rotate* instructions in both instruction sets, although the Z80's are more complex than those of the 6502. In general, their effect is to shift each bit in a register one position to the right or to the left. They differ in detail in their treatment of the end bits of the register — a bit must be shifted out of the register at one end while another bit is shifted in at the other end. If bit 7 is shifted out of the register and put immediately back into bit 0, then the operation is a *rotate left.* If bit 0 is shifted into bit 7 the operation is a *rotate right.* If this is done, then the contents of the register change in order, no new values are introduced, and after eight such rotations the register will be restored to its original state.

If rotation is not employed, then a destination for the shifted-out bit is necessary, and a source must be found for the shifted-in bit. Both are most often supplied by the various condition flags of the processor status register (PSR), and in particular the carry flag. In constructing a multiplication subroutine to multiply two single-byte numbers, we need to shift the multiplicand left and the multiplier right. The multiplicand bits must be shifted out into the hi-byte of the multiplicand while zeros are shifted into the unoccupied bits. The multiplier bits have to be shifted through a PSR flag for testing, but their destination, and the state of the shifted-in multiplier bits, is unimportant unless we need to preserve the contents of the multiplier. All that concerns us about the multiplier during multiplication is whether the shifted-out bit is one or zero.

Given, therefore, that the multiplier is stored at address MPR, the multiplicand at MPDLO, and the product at PRODLO and PRODHI, we can write these subroutines as follows:

EIGHT-BIT MULTIPLICATION

6502			Z80		
	ORG	$C100		ORG	$D000
START	LDA	#$00	START	LD	BC,(MPR)
	STA	PRODLO		LD	B,$08
	STA	PRODHI		LD	DE,(MPDLO)
	STA	MPDHI		LD	D,$00
	LDX	#8		LD	HL,$00
	CLC		LOOP0	SRL	C
LOOP0	ROR	MPR		JR	NC,CONT0
	BCC	CONT0		CALL	ADDIT
	JSR	ADDIT	CONT0	SLA	E
CONT0	ASL	MPDLO	ENDLP0	DJNZ	LOOP0
	ROL	MPDHI		LD	PRODLO
	DEX			RTS	
ENDLP0	BNE	LOOP0	MPR	DB	$E2
	RTS		MPDLO	DB	$7A
MPR	DB	$E2	MPDHI	DB	$00
MPDLO	DB	$7A	PRODLO	DW	$0000
MPDHI	DB	$00	ADDIT	ADD	HL,DE
PRODLO	DB	$00		RET	
PRODHI	DB	$00			
ADDIT	CLC				
	LDA	PRODLO			
	ADC	MPDLO			
	STA	PRODLO			
	LDA	PRODHI			
	ADC	MPDHI			
	STA	PRODHI			
	RTS				

As can be seen from this example, programming the Z80 is made much easier by its 16-bit registers and associated instructions. In particular, compare the ADDIT subroutine in the two programs. The 6502 version uses ROR to rotate the multiplier rightwards through the carry, and ASL and ROL to shift the multiplicand leftwards out of MPDLO into MPDHI through the carry. The loop is controlled by the X register as a counter.

The Z80 version uses SRL to shift the multiplier rightwards through the carry, and SLA and RL to shift the multiplicand leftwards in DE via the carry. The loop is controlled by register B as a counter. Notice that the ADD instruction not only supports 16-bit register arithmetic, but also is not affected by the carry flag — unlike ADC.

In the next instalment of the course, we will discuss methods of division, and consider various ways of controlling the screen display. This will complete the tutorial element of the course, and will be followed by 6502 and Z80 exercises and examples in future instalments.

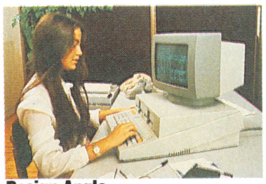
Exercise 15

1) Write a multiplication subroutine using a 16-bit multiplicand and an eight-bit multiplier of your choice.

2) Multiplication is merely repeated addition: write an eight-bit by eight-bit multiplication subroutine that does not use the shift or rotate instructions.

ITALIAN ELEGANCE

In the highly competitive world of office machinery, the market is dominated by the American and Japanese giants. But one European-based multinational,Olivetti, has gained such a reputation as a manufacturer of 'elegant and serious' office machines that it is now considered a major force in business automation.

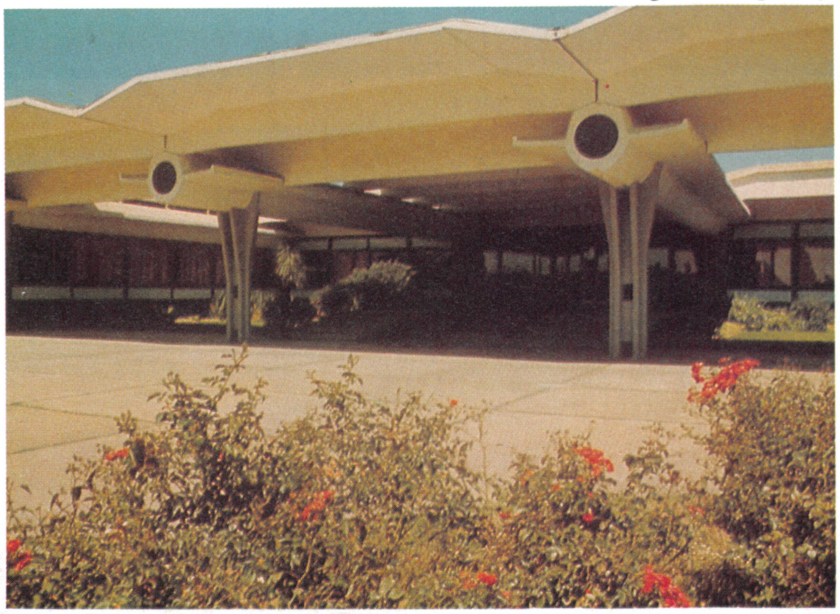

Design Angle
The Olivetti M20 business computer is a 16-bit machine that was first launched in 1981. It incorporates a Zilog Z8000 microprocessor and uses the Olivetti PCOS operating system. Olivetti has recently launched an IBM PC-compatible version of this machine

Smart Exterior
Olivetti has always been noted for the advanced design of its products. But its concern for smart appearances also applies to the architecture of company buildings. This office was erected in 1959 and shows many features that were subsequently adopted by other architects

Camillo Olivetti founded his company in 1908. With a workforce of 20, he set up a machine shop in Ivrea, then a small town in rural Northern Italy, and began production of the company's first typewriter, the M1. At that time the Italian economy was still predominantly agricultural, lacking the heavy industry that had fuelled expansion in Germany, Britain and the USA. Despite this, Olivetti's production rose steadily, growing from four machines per day in 1914 to 50 per day in 1929.

In the 1930s, Camillo's son Adriano began to reorganise the company structure, introducing students from Olivetti's own night school, founded in 1924. Company housing and social benefits were also made available — a 'cradle to the grave' policy reminiscent of that later used to great effect by the Japanese. While industry in the rest of the world struggled through the pre-war economic depression, Olivetti continued to grow, and by 1933 the company had sold 15 million office products. In 1937 the first Olivetti teleprinter was launched, followed in 1940 by the company's first calculator.

The Second World War brought a temporary halt to Olivetti's expansion, but in the post-war years the company concentrated on developing new markets, arriving in the UK in 1948. The company's success was founded on elegant design and quality products, and even IBM executives were forced to admit that Olivetti products 'fitted together like a beautiful picture puzzle'.



In the 1950s and 1960s, Olivetti began to concentrate on the development of business computers. This process began with the introduction of a numeric accounting machine in 1955, and Olivetti's first mainframe computer — the Elea — was developed a few years later.

The company continued to diversify, moving away from mechanical office machinery towards the electronics-based equipment that Olivetti identified as the major trend in office automation. A new range of minicomputers was introduced, together with banking terminals and communications equipment.

Today, Olivetti manufactures a vast range of electronic business machines — over 1,000 different products are available in the UK alone — and the company invests large sums in developing software support for its machines. In 1982, Olivetti was the second largest computer manufacturer in Europe (surpassed only by IBM), with the M10 portable computer and the M20 business machine both selling well.

The M10 hand-held computer weighs about 1.7 kg (3.25 pounds) and comes equipped with an 8×40 character screen. The machine is battery-powered and has 8 Kbytes of internal RAM that can be expanded to 64 Kbytes. The M20 16-bit business machine runs on a Z8001 micro-processor, which has proved unpopular with other 16-bit machine manufacturers. It also contains an 8086 processor, allowing some compatibility with CP/M-86 and MS-DOS.

Olivetti is also planning a new IBM-PC compatible machine that the company claims will cost less than its IBM rival. Called the M24, it features an 8086-2 processor and has the option of a Z8001 card to make it compatible with the Olivetti M20. This compatibility has meant that Olivetti has been forced to abandon its own PCOS operating system.

Olivetti has recently signed a deal with AT&T (the world's largest telecommunications company) to collaborate on a project to develop the Unix operating system. And there is no doubt that in the future Olivetti's worldwide dealer network and productive research and development division will maintain the company's reputation for stylish and well-manufactured business products.

Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

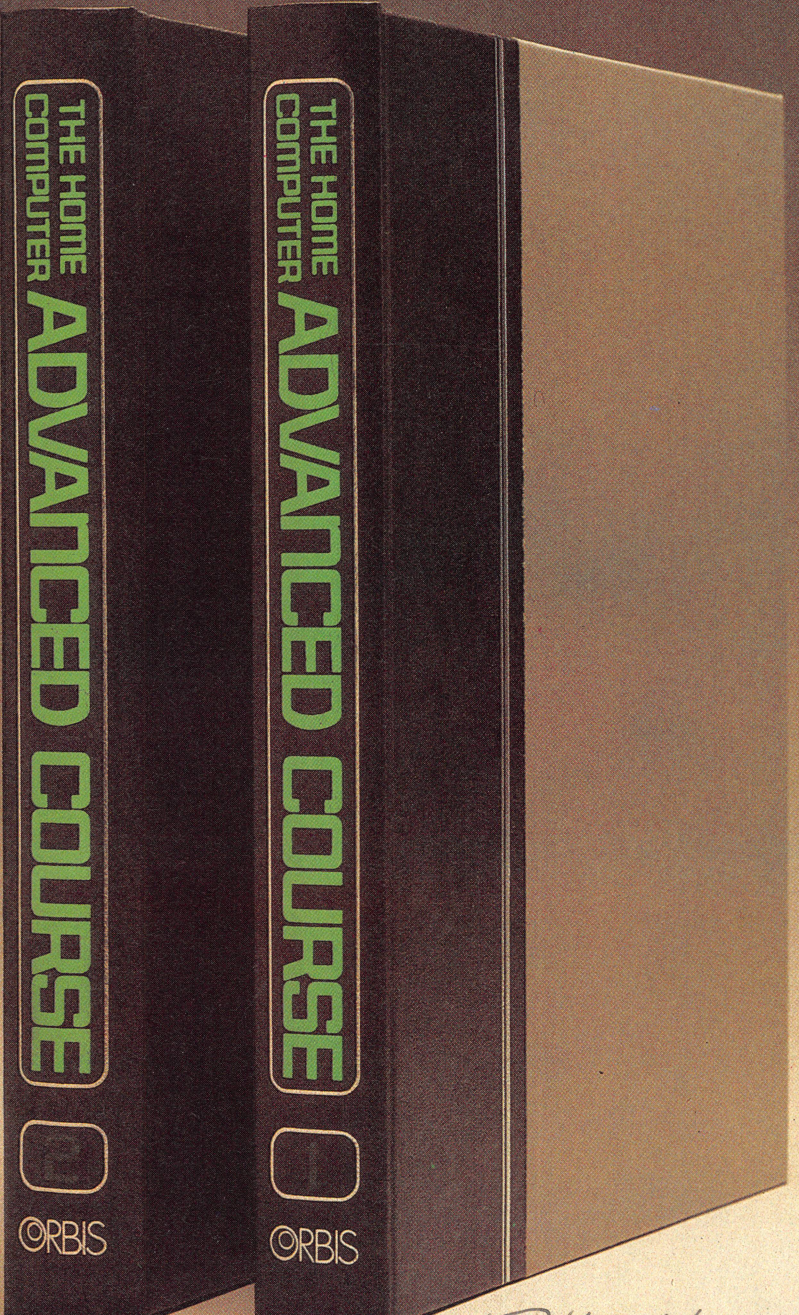
And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

sinclair

WE HAVE DESIGNED BINDERS SPECIALLY TO KEEP YOUR COPIES OF THE 'ADVANCED COURSE' IN GOOD ORDER.


THE HOME COMPUTER ADVANCED COURSE
2
ORBIS
THE HOME COMPUTER ADVANCED COURSE
1
ORBIS

All you have to do is complete the reply-paid order form opposite – tick the box and post the card today – **no stamp necessary!**

By choosing a standing order, you will be sent the first volume free along with the second binder for £3.95. The invoice for this amount will be with the binder. We will then send you your binders every twelve weeks – as you need them.

Important: This offer is open only whilst stocks last and only one free binder may be sent to each purchaser who places a Standing Order. Please allow 28 days for delivery.
Overseas readers: This free binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain binders now. For details please see inside the front cover. Binders may be subject to import duty and/or local tax.
The Orbis Guarantee: If you are not entirely satisfied you may return the binder(s) to us within 14 days and cancel your Standing Order. You are then under no obligation to pay and no further binders will be sent except upon request.

[illegible]

PLACE A STANDING ORDER TODAY.